MASTER EN SCIENCES INFORMATIQUES

Algorithmes génétiques et recuit simulé

Pierre, Anne-Carine

Award date:
1991

Awarding institution:
Universite de Namur

ALGORITHMES GENETIQUES

ET

RECUIT SIMULE

PIERRE Anne-Carine

Année académique 1990-1991

Nous remercions très vivement Monsieur FICHEFET,
promoteur de ce mémoire et Monsieur LECLERCQ
qui ont contribué à l'élaboration de ce travail

TABLE DES MATIERES

RESUME : Dans la vie quotidienne, l'individu est souvent confronté à des problèmes d'optimisation. Le problème du voyageur de commerce en est un exemple bien connu. Deux techniques de recherche permettent de résoudre efficacement de tels problèmes : les méthodes dites *fortes* et les méthodes dites *faibles*. Les méthodes fortes sont basées sur l'utilisation d'heuristiques pour orienter la recherche. Ces heuristiques sont néanmoins très difficiles à établir; c'est pourquoi on fera généralement appel aux méthodes faibles. Celles-ci n'utilisent que très peu d'informations dépendant de l'application mais, en général, elles ne sont pas très performantes. Cependant, les algorithmes génétiques et le recuit simulé sont des méthodes faibles très efficaces car elles sont basées sur les phénomènes naturels suivants : l'évolution naturelle et la thermodynamique. Toutefois, ces méthodes ont tendance à converger vers une solution sous-optimale. Différents remèdes à ce problème ont été envisagés pour atteindre une solution très proche de l'optimum.

ABSTRACT : In everyday, an individual is often confronted with optimization problems. The travelling salesman problem is a well-known example of them. Two search techniques are able to solve such problems efficiently : the so-called strong methods and the so-called weak methods. Strong methods are based on the use of heuristics to guide the search. Nevertheless these heuristics are very hard to find. That's why weak methods are usually used. These weak methods use as little domain-dependent knowledge as possible but they are generally not very efficient. However, genetic algorithms and simulated annealing are weak methods much more efficient because they are modelled on the following processes found in nature : natural evolution and thermodynamics. Unfortunately, these methods tend to converge towards a sub-optimal solution. Many remedies to tackle this problem have been proposed to reach a very nearly optimal solution.

INTRODUCTION

La maîtrise de systèmes dynamiques complexes (par exemple, les volumes macroscopiques d'un gaz, les fourmilières et le développement génétique de populations d'organismes) se réduit souvent à une simple optimisation d'une fonction numérique.

De tels systèmes disposent d'un comportement auto-organisationnel. Les fourmis, par exemple, peuvent s'organiser entre-elles pour établir un chemin entre une source de nourriture et la fourmilière. Il n'y a pas de superviseur central qui décide de l'endroit où les fourmis doivent établir leur chemin puisque ce chemin provient des interactions locales entre les fourmis. Le développement d'une population d'organismes présente également un comportement auto-organisationnel. L'absence de superviseur central permet à l'organisme de s'adapter d'une manière optimale à un environnement complexe. Le comportement auto-organisationnel a l'avantage que tout élément pris isolément n'affecte pas les performances de tout le système; si on retire une fourmi d'une fourmilière, on ne modifie pas pour autant le comportement global de celle-ci.

D'autre part, le comportement des systèmes dynamiques complexes est également flexible : les fourmis contourneront un obstacle qui aurait été placé sur leur chemin. Une population d'organismes peut donc se développer en s'adaptant elle-même à tout changement de l'environnement. Par conséquent, tout système dynamique complexe peut être résolu en faisant appel à sa propre dynamique.

Le développement d'une population d'organismes a servi de modèle pour l'élaboration de la puissante technique de recherche mise au point par John HOLLAND et connue sous le nom d'algorithmes génétiques (AG).

Par la publication de "*Adaptation in Natural and Artificial Systems*" en 1975, HOLLAND démontra que de simples transformations peuvent améliorer considérablement des structures et qu'une population de chaînes de bits peut se développer de la même manière que le font certains animaux. Il a pu mettre en évidence que les algorithmes génétiques tendent à converger vers des solutions quasi-optimales.

Une autre technique s'inspirant des concepts de la physique statistique a été mise au point par KIRKPATRICK en 1983 et porte le nom de "Recuit Simulé". Il fut le premier a proposer et à démontrer l'application de cette technique pour trouver des solutions de coût minimal à des problèmes d'optimisation tels que le routage des circuits intégrés (VLSI) et imprimés.

Le premier chapitre nous donne une description et une analyse des différentes étapes (initialisation, sélection, variation) constituant un algorithme génétique. L'équivalent du croisement, de l'inversion et de la mutation de matériaux chromosomiques y sont également présentés.

Le deuxième chapitre traite de l'optimisation des paramètres et définit les différents opérateurs utilisés pour répondre au problème du voyageur de commerce.

Le chapitre III donne des indications pour la détermination des paramètres de contrôle des AG et expose le problème de la convergence prématurée et la façon d'y remédier.

Le chapitre IV présente la méthode du recuit simulé en relation avec la physique statistique et le risque de se retrouver figé dans des minima locaux.

Le chapitre V reprend le compte-rendu des essais réalisés personnellement pour résoudre le problème du voyageur de commerce (PVC) par les deux méthodes étudiées. On peut également y trouver un listing des programmes que j'ai conçus pour mener à bien ces expériences. Ces essais ont permis de déterminer les différentes valeurs à donner aux paramètres pour atteindre une solution optimale quelle que soit la complexité du problème (voir tableau figure 5.1 pour le recuit simulé et tableau figure 5.6 pour les algorithmes génétiques). Comme résultats de l'exécution de ces deux logiciels, on présente les configurations initiales et finales d'un PVC à 100 villes pour le recuit et à 70 villes pour les algorithmes génétiques.

L'annexe 1 rappelle les notions de biologie relatives à la génétique.

Notons que le sigle [n°] signifie un renvoi à la bibliographie.

CHAPITRE I : DESCRIPTION ET ANALYSE DES ALGORITHMES GENETIQUES

I.1 L'EVOLUTION COMME RECHERCHE

Tout organisme est un amalgame de caractéristiques déterminées par les gènes de ses chromosomes. Chaque gène a de multiples formes possibles, les allèles, qui déterminent ses différentes caractéristiques. Certains pois par exemple, ont un seul gène qui détermine la couleur de la fleur, un allèle étant responsable de la couleur blanche de cette fleur, l'autre de la couleur rose. Plusieurs gènes peuvent aussi déterminer conjointement une caractéristique donnée, par exemple, la couleur de l'oeil pour les humains est déterminée par deux gènes. Il existe donc plusieurs allèles pour le gène transmettant la couleur de l'oeil. Certains gènes auront jusqu'à cent allèles.

L'ensemble de tous les gènes d'un organisme constitue son génotype. Les génotypes du système sont donc toutes les structures génétiques possibles qui peuvent être créées par différentes combinaisons d'allèles. Le caractère d'un organisme (et par conséquent de son génotype) permet de mesurer ses performances d'après son aptitude à s'adapter à l'environnement dans lequel il vit. Tout génotype sera donc évalué selon son caractère qui représente sa capacité à se maintenir en vie (c'est-à-dire son aptitude à lutter pour obtenir de la nourriture, un espace vital...) et à engendrer des descendants.

L' *évolution* est un processus qui modifie, génération après génération, le bagage génétique des organismes pour mieux les adapter à leur environnement. Une génération correspondra à une itération de l'algorithme. Plus techniquement, l'évolution est une recherche de points dans un domaine multidimensionnel de combinaisons de gènes - les génotypes.

La grande quantité de génotypes possibles est une des premières difficultés auxquelles cette recherche sera confrontée. Un génotype de 10000 gènes par exemple (une quantité modérée pour un vertébré), dont chaque gène ne comporte que deux allèles, possède 2^{10000} formes possibles.

La réelle complexité vient toutefois de l'interaction entre gènes. A cause de cette interaction, l'effet d'un allèle peut dépendre fortement de la nature des autres allèles présents. Il est toutefois possible que des génotypes à peu près identiques aient des caractères très différents. Dès lors, l'évolution devient surtout une recherche d'ensembles d'allèles de différents gènes bien adaptés l'un à l'autre, qui augmentent fortement la taille du génotype.

L'idée des AG est d'extraire et de généraliser des processus critiques de l'évolution et de les utiliser pour résoudre d'autres problèmes de recherche.

I.2 LES PROCESSUS DE L'EVOLUTION

Les organismes d'une population sont très diversifiés. Sans cette diversité, la vie n'aurait pas pu se développer. Le terme de variation est utilisé pour désigner le processus par lequel la progéniture d'un organisme se différencie de cet organisme. La variation agit en introduisant des changements dans le matériel héréditaire - les génotypes - de la progéniture.

Certains organismes ont des caractéristiques qui les aident à mieux s'adapter que d'autres à leur environnement; les plus grands auront le plus de chances de survivre et de se reproduire que les plus petits. Ce processus est connu sous le nom de selection naturelle.

En d'autres mots, la variation introduit de nouveaux organismes et la sélection naturelle favorise la dispersion des organismes les plus utiles. Ensemble, ces deux processus sont très efficaces pour engendrer des organismes qui sont de mieux en mieux adaptés à leur environnement.

Un exemple récent est la propagation du mélanisme dit industriel parmi beaucoup d'espèces de papillons de nuit. La variété "normale" de papillons est légèrement pigmentée. Dans beaucoup de régions industrielles où la végétation est noircie par la pollution, ces papillons de nuit n'auront pas une grande chance de survivre car ils seront plus exposés à la prédation des oiseaux. Le processus de variation introduira occasionnellement des papillons de nuit à la pigmentation noirâtre et la sélection naturelle répandra

cette variété. Comme conséquence, la variété noire a presque complètement remplacé celle légèrement pigmentée dans certaines régions.

I.3 QUELQUES FORMALISATIONS

Pour utiliser l'évolution comme une technique de recherche, des abstractions doivent être faites en ce qui concerne sa structure et ses processus.

1. Les chomosomes ou génotypes (ou encore structures) se présentent sous la forme de chaînes de longueur fixe ayant un nombre fini de valeurs possibles ou allèles en chaque position. Le premier élément de la chaîne est l'allèle du premier gène, le second élément est l'allèle du second gène, etc... .

2. La longueur n des génotypes est maintenue constante et une population contiendra un nombre fini de génotypes.

3. Chaque génotype a un caractère ou une aptitude à survivre et à se reproduire.

Désignons par α l'ensemble de tous les génotypes possibles :

$\alpha = \{ a_1 a_2 \ldots a_n \mid a_i$ est un allèle du gène i $\}$

L'application caractère μ affecte une valeur positive au caractère de chaque génotype de α :

$$\mu : \alpha \text{ -----> } R^+$$

Un AG modifie une population de génotypes pas à pas. La population au temps t est désignée par $\beta(t)$. La taille de la population est maintenue constante à M génotypes.

Pour pouvoir observer le comportement d'ensembles d'allèles adaptés entre eux et de combinaisons d'allèles en général, on définit des sous-ensembles de α ayant des allèles en commun. Pour réaliser ceci, le symbole [] indique que

nous ne nous intéressons pas au type d'allèle qui survient en une position donnée d'un génotype. Donc, $a_1[]a_3[]$ ... [] désigne le sous-ensemble de tous les éléments de α qui ont l'allèle a_1 pour premier gène et l'allèle a_3 pour troisième gène. D'où, $a_1a_2a_3$ ...a_n est un élément de cet ensemble, mais $a_1a_2b_3a_4$ ... a_n n'en est pas un. Les sous-ensembles de α qui ont des allèles en commun de cette manière sont appelés des schémas. Un génotype $A \in \alpha$ qui appartient à un schéma ξ est appelé une instance de ce schéma.

I.4 ESQUISSE D'UN ALGORITHME GENETIQUE

Un algorithme génétique est une procédure itérative qui maintient constante la taille d'une population P(t) de solutions candidates. A chaque itération, appelée une génération, la structure de la population courante est évaluée et, sur base de ces évaluations, une nouvelle population P(t+1) de solutions candidates est formée.

La population initiale P(0) peut être choisie à l'aide d'heuristiques ou simplement au hasard. Les structures de la population P(t + 1) sont choisies à partir de P(t) par un processus de sélection aléatoire qui garantit qu'une structure est choisie un nombre de fois approximativement proportionnel aux performances relatives de cette structure par rapport au reste de la population.

Squelette d'un algorithme génétique.

```
t <--- 0;
initialiser P(t);  -- P(t) est la population au temps t --
évaluer P(t);
Tant que (condition de terminaison non satisfaite) faire
début
      t <--- t + 1;
      sélectionner P(t) ;
      appliquer des opérateurs génétiques à P(t);
      évaluer P(t);
Fin
```

Les étapes d'un AG sont :

1. Initialisation : Faire t=0 et choisir au hasard M génotypes de α pour former $\beta(0)$

2. Sélection : Reproduire les génotypes $A \in \beta(t)$ en fonction de leur caractère $\mu(A)$, formant $\beta'(t)$.

3. Variation : Modifier les génotypes de $\beta'(t)$ en utilisant les opérateurs génétiques pour produire la nouvelle population $\beta(t+1)$.

4. Faire t=t+1 et retourner à l'étape 2.

Il s'ensuit de l'esquisse de l'algorithme qu'un AG alterne la sélection et la variation, les deux processus de l'évolution. L'étape d'initialisation n'a pas d'équivalent dans l'évolution naturelle mais elle est obligatoire dans un AG pour lancer le processus de recherche.

Durant chaque itération de l'algorithme - une génération - le caractère de chaque chaîne est évalué et les chaînes sont choisies au hasard pour devenir "les parents" et se reproduire en fonction de leur caractère.

Les nouvelles chaînes subissent alors l'étape de variation qui utilise les opérateurs de croisement, d'inversion et de mutation pour ensuite remplacer les chaînes "parents". Une génération suivante peut à nouveau commencer.

La séquence des populations P(t) générées par l'algorithme correspond à la trajectoire de la recherche dans l'espace de toutes les chaînes possibles.

I.5 SELECTION

Cette étape d'un AG doit reproduire chaque génotype selon son caractère à la condition que la population intermédiaire $\beta'(t)$ ait à nouveau M génotypes. Comme indiqué à la figure 3, chaque $A_i \in \beta(t)$ a une chance

$$\frac{\mu(A_i)}{\sum_{j=1}^{M} \mu(A_j)}$$

de prendre une des M places de la population $\beta'(t)$.

Sélection :

Pour k:=1 jusque M faire

< Choisir un génotype $A_i \in \beta(t)$ pour devenir le k-ième génotype de $\beta'(t)$.

Chaque $A_i \in \beta(t)$ a une chance $\frac{\mu(A_i)}{\sum_{j=1}^{M} \mu(A_j)}$ d'être sélectionné >

Figure 3 : L'étape de sélection d'un AG.

En définitive, l'étape de sélection "reproduit" les génotypes selon leur caractère. Par exemple, si A_i et A_j sont des génotypes de la population $\beta(t)$ et que $\mu(A_i) = 2\ \mu(A_j)$, alors A_i aura approximativement deux fois plus de descendants que A_j. En général, le nombre prévisible de la progéniture d'un génotype $A_i \in \beta(t)$ est

$$M \frac{\mu(A_i)}{\sum_{j=1}^{M} \mu(A_j)}$$

(Par exemple si $\frac{\mu(A_i)}{\sum \mu(A_j)} = \frac{1}{2}$ alors, A_i sera choisi $M*1/2$ fois).

Nous pouvons encore écrire ceci sous la forme

$$M. \frac{\mu(A_i)}{\Sigma \mu(A_j)} = \frac{\mu(A_i)}{\frac{\Sigma \mu(A_j)}{M}} = \frac{\mu(A_i)}{\bar{\mu}(t)}$$

où $\bar{\mu}(t) = \Sigma \mu(A_j) / M$ désigne le caractère moyen de la population $\beta(t)$.

La quantité $\mu(A_i) / \bar{\mu}(t)$ peut être considérée comme le caractère "normalisé" de A_i , l'utilité de A_i étant mesurée en fonction du caractère moyen des autres membres de la population.

Etant donné l'importance de l'interaction entre gènes, nous avons vu que l'évolution est une recherche de combinaisons réussies d'allèles. Nous avons également défini les schémas comme des sous-ensembles de α ayant des combinaisons d'allèles en commun. Voyons à présent ce qu'il advient d' une combinaison d'allèles formant les génotypes de la population pendant l'étape de sélection, ou ce qui arrive à un schéma ξ avec une ou plusieurs instances de $\beta(t)$.

Soit $M\xi(t)$ instances d'un schéma ξ de $\beta(t)$. Chaque $A_i \in \xi$ de $\beta(t)$ produira approximativement $\mu(A_i) / \mu(t)$ descendants. Dès lors, le nombre total de progénitures d'instances de ξ de $\beta(t)$ est égal à

$$\sum_{A_i \in \xi} \frac{\mu(A_i)}{\bar{\mu}(t)} = \frac{\frac{\Sigma \mu(A_i)}{M\xi(t)}}{\bar{\mu}(t)} . M\xi(t) = \frac{\bar{\mu}\xi(t)}{\bar{\mu}(t)} . M\xi(t)$$

où $\bar{\mu}\xi(t)$ est le caractère moyen des instances de ξ dans $\beta(t)$.
Si $M'\xi(t)$ est le nombre d'instances de ξ dans $\beta'(t)$, nous avons

$$M'\xi(t) = \frac{\bar{\mu}\xi(t)}{\bar{\mu}(t)} . M\xi(t)$$

Chaque schéma avec des instances dans $\beta(t)$ aura donc des instances dans $\beta'(t)$ proportionnellement au nombre moyen de ses instances dans $\beta(t)$. Puisque chaque génotype est une instance de 2^n schémas (vu que chaque génotype est une instance de n'importe quel schéma obtenu en substituant un ou plusieurs de ses n gènes par des []) et qu'il y a M génotypes dans une population, il y aura environ entre 2^n et $M.2^n$ schémas qui seront traités de cette manière. Cette manipulation d'un très grand nombre de schémas par des opérations sur relativement peu (M) génotypes est appelée *parallélisme intrinsèque.*

I.6 VARIATION

Comme nous l'avons vu, l'étape de variation sert à tester des combinaisons d'allèles dans de nouveaux contextes. En termes de schéma, cela signifie que l'étape de variation doit générer de nouvelles instances de schémas existants.

Bien qu'il y ait entre 2^n et $M.2^n$ schémas avec des instances dans la population, ceci ne constitue en fait qu'une petite fraction du nombre total de schémas possibles. Si chaque gène n'a que deux allèles, il y aura 3^n schémas. C'est pourquoi, la deuxième fonction de l'étape de variation est de générer des instances de "nouveaux" schémas, c'est-à-dire des schémas n'ayant pas d'instances dans la population.

I.6.1 L'OPERATEUR DE CROISEMENT

L'opérateur de croisement est l'opérateur génétique le plus important. En biologie, le crossing-over est un processus produisant des combinaisons d'allèles par l'échange de segments entre des paires de génotypes. Dans un AG, le crossing-over est une manière de construire deux nouveaux génotypes - la progéniture - à partir de deux génotypes "parents" (figure 4).

Croisement appliqué à $A = a_1 a_2 \ldots a_n$ et $B = b_1 b_2 \ldots b_n$:

1. Un nombre x est choisi au hasard dans {1,2,...,n-1}.

2. Deux nouveaux génotypes sont formés à partir de A et de B en échangeant les ensembles d'allèles situés à droite de la position x, ce qui nous donne

   $C = a_1 \ldots a_x \; b_{x+1} \ldots b_n$ et $D = b_1 \ldots b_x \; a_{x+1} \ldots a_n$

Figure 4 : L'opérateur de croisement

Les deux effets directs de l'opérateur de croisement sont :

1. La génération de nouvelles instances de schémas existants : A est une instance du schéma $a_1 \; a_2$ []...[] et C est une nouvelle instance de ce schéma (en supposant $a_i \neq b_i$ pour tout $i \geq x$ et que C n'est pas déjà un membre de la population).

2. La génération d'instances de nouveaux schémas :
   Le schéma []...[] $a_x \; b_{x+1}$ []...[] a une instance C, bien que ni A ni B ne soient des instances de celui-ci (en supposant $a_x \neq b_x$ ou $a_{x+1} \neq b_{x+1}$).

Chaque croisement affecte un grand nombre de schémas : Supposons que A diffère de B sur y positions à gauche de x et sur z positions à droite de x. Après le crossing-over, tout schéma défini en une ou plusieurs positions parmi ces y positions et en une ou plusieurs positions parmi ces z positions, n'aura ni A ni B comme instance. Sur la gauche, il y a $2^y - 1$ manières de combiner un ou plusieurs des y allèles avec des []; De même, il y a $2^z - 1$ manières sur la droite; Pour les n-(y+z) autres positions, on peut avoir soit un allèle soit un [].
Donc, il y a

$$(2^y - 1)(2^z - 1)(2^{n-(y+z)}) = 2^n - 2^{n-y} - 2^{n-z} + 2^{n-(y+z)}$$

schémas qui admettent C et D comme instances mais pas A et B.

L'opérateur de croisement est très puissant car il peut combiner en une opération des allèles ou ensembles d'allèles de différents génotypes. La figure 5 présente l'opérateur de croisement comme une partie de l'étape de variation d'un AG. Notons qu'il n'y a qu'une proportion P_c des génotypes de $\beta'(t)$ qui subit l'opérateur de croisement et que ces nouveaux génotypes remplacent leurs génotypes "parents".

Variation :
Croisement :
  Pour k:=1 jusque (M div 2) faire
    <Choisir au hasard deux génotypes A_i et A_j dans $\beta'(t)$>
    <Appliquer l'opérateur de croisement à A_i et A_j avec
      la probabilité P_c >
    Si < L'opérateur de croisement est appliqué > alors
      < Remplacer A_i et A_j par les nouveaux génotypes >

Figure 5 : L'étape de variation avec l'opérateur de croisement

Nous avons vu que l'étape de sélection disperse les ensembles d'allèles dans la population selon leur "utilité" c'est-à-dire

$$M'_{\xi}(t) = \frac{\bar{\mu}_{\xi}(t)}{\bar{\mu}(t)} . M_{\xi}(t)$$

Définissons la "longueur" d'un schéma. Soit $i_1 < i_2 < ... < i_h$, les positions définies (celles n'ayant pas un []) d'un schéma ξ. La longueur de ξ est désignée par $l(\xi) = i_h - i_1$, par exemple $l([][]a_3\ []a_5[][]a_8[]...[]) = 8 - 3 = 5$.

Si on soumet une instance d'un schéma ξ à un croisement situé en dehors des positions définies de ξ, un des génotypes résultants devient une instance de ce schéma. Par exemple, si $a_1a_2 ...a_n$ est croisée avec $b_1b_2 ...b_n$ en x=1, alors $b_1a_2a_3 ...a_n$ est à nouveau une instance

de [][]a_3[]a_5[][]a_8[]...[]. Puisque les génotypes résultants remplacent leurs parents, $M_{\xi}^{\cdot}$ (t) ne sera pas modifié par un croisement sur une instance de ξ si celui-ci est appliqué en dehors des positions définies de ce schéma.

Si une instance de ξ est croisée avec une autre instance de ξ, les génotypes résultants seront aussi des instances de ξ, même si le point de croisement tombe à l'intérieur des positions définies de ξ .

Dans tous les autres cas, nous supposerons que chaque croisement d'une instance de ξ, tombant à l'intérieur des positions définies de ξ, est destructif, c'est-à-dire qu'aucun des résultants n'est une instance de ξ. Ceci est une surestimation car si nous croisons $a_1a_2a_3$ ...a_n avec $b_1b_2a_3b_4$ ...b_n en x=3 (un point tombant à l'intérieur des positions définies de []a_3[]a_5[][]a_8[]...[]), nous obtenons un génotype $b_1b_2a_3a_4$...a_n qui est à nouveau une instance de []a_3[]a_5[][]a_8[]...[]. Nous pouvons maintenant démontrer un théorème fondamental des AG, le théorème du schéma qui dit que l'importance des combinaisons d'allèles réussies n'est qu'un peu perturbée par l'étape de variation.

I.6.2 L'OPERATEUR D'INVERSION

L'opérateur d'inversion modifie la position des gènes sur un génotype. Il permet aux allèles adaptés l'un à l'autre de se regrouper sur un génotype afin d'augmenter leur probabilité de se déplacer ensemble durant le croisement. Ceci augmente les performances d'un AG puisque les bonnes combinaisons d'allèles se répandront plus rapidement dans la population.

Jusqu'ici, la position d'un gène sur un génotype est fixe. Dès lors, si nous désirons modifier la position des gènes, les gènes doivent garder la même signification fonctionnelle dans n'importe quelle autre position (comme c'est le cas en biologie). La manière la plus simple de réaliser ceci est d'affecter à chaque allèle un indice qui indique à quel gène il appartient. Chaque allèle est alors représenté par une paire (i,a) qui signifie que a est

un allèle du i-ème gène. Un génotype $A = a_1a_2...a_n$ peut donc être représenté par toute permutation de $(1,a_1)(2,a_2)...(n,a_n)$, par exemple $(3,a_3)(2,a_2)(1,a_1)(4,a_4)...(n,a_n)$.

La première représentation de A est une instance de $(1,a_1)[][](4,a_4)[]...[]$ et la deuxième, une instance de $[][](1,a_1)(4,a_4)[]...[]$. Si les allèles a_1 et a_4 forment une bonne combinaison, ils se répandront d'autant plus rapidement dans la population s'ils appartiennent à des instances du second schéma, qui est le plus court. Puisque la vitesse de reproduction d'instances de schémas dépend de la longueur du schéma, on recherche constamment les schémas les plus courts. L'inversion favorise cette recherche des schémas les plus courts.

Inversion appliquée à $A = a_1a_2 ...a_n$ et $B = b_1b_2...b_n$ (où chaque allèle a_i est maintenant une paire (i,a)) :

1. Choisir au hasard deux nombres x'_1 et x'_2 dans {0,1,2,...,n+1} et définir $x_1 = \min(x'_1,x'_2)$ et $x_2 = \max(x'_1,x'_2)$.

2. Former un nouveau génotype à partir de A en intervertissant le segment qui se trouve à droite de la position x_1 et à gauche de la position x_2, ce qui donne

$$B = a_1...a_Xa_{X-1}a_{X-2}...a_{X+1}a_X...a_n$$

Figure 6 : L'opérateur d'inversion

L'opérateur d'inversion, comme l'opérateur de croisement, agira le plus souvent sur de longs schémas. Dès lors, les changements dans les ensembles d'allèles seront concentrés dans les schémas les plus longs, où ces changements sont souhaitables.

Une restriction doit être faite en ce qui concerne l'opérateur de croisement quand il est utilisé en combinaison avec l'inversion. A cause de

l'inversion, les allèles de deux génotypes de $\beta'(t)$ n'auront pas toujours la même position pour un gène donné. L'opérateur de croisement peut donc produire des génotypes avec deux (ou plus) allèles pour un gène donné, ou des génotypes sans aucun allèle pour un gène donné. Par exemple, croiser $(1,a_1)(2,a_2)(3,a_3)...(n,a_n)$ avec $(1,b_1)(3,b_3)(2,b_2)$ ... (n,b_n) au point x=2 produit $(1,a_1)(2,a_2)(2,b_2)...(n,b_n)$. La manière la plus simple de remédier à ceci est de n'autoriser un croisement qu'entre des représentations où les gènes sont indicés dans le même ordre. Pour cela, il faut que la probabilité d'inversion P_I soit petite afin d'avoir assez de génotypes pour laisser l'opérateur de croisement s'exécuter.

Variation :
Inversion :
Pour k:=1 jusque M faire
< Choisir au hasard un génotype A_i dans β' >
< Appliquer l'opérateur d'inversion à A_i avec la probabilité P_I >
Si < L'inversion est appliquée > alors
< Remplacer A_i par le nouveau génotype >

Figure 7 : L'étape de variation avec l'opérateur d'inversion.

Malgré son utilité évidente, l'opérateur d'inversion est rarement utilisé dans un AG à cause du temps de charge de l'unité centrale. Une autre explication possible est que des instances de longs schémas ont beaucoup de chances de survivre à l'étape de variation si le seul opérateur est le croisement puisque tout génotype n'est pas soumis à un croisement (P_C se situe habituellement dans la marge des 30 à 60 %) et que de longs schémas reçoivent des instances d'autres sources que leurs propres instances.

La fonction de l'opérateur d'inversion n'est pas seulement d'écourter des schémas mais aussi, tout comme l'opérateur de croisement, de générer des instances de schémas existants et de nouveaux schémas si chaque gène a le même groupe d'allèles. Si le segment d'un génotype A diffère de ce

même segment interverti sur y positions, il existera alors 2^{n-y} schémas qui admettrons comme instance le génotype obtenu (chaque schéma qui a un [] aux y différentes positions et soit un allèle soit un [] aux autres positions). Les $2^n - 2^{n-y}$ schémas qui restent sont de nouveaux schémas.

Contrairement à l'opérateur de croisement, l'opérateur d'inversion offre une plus grande chance de survie aux instances des plus longs schémas. Par exemple, si une instance du schéma $a_1[]...[]a_n$ est croisée, il y a peu de chance que le résultat soit une instance de ce schéma, alors que le résultat de l'inversion est très souvent une instance du schéma.

Malgré cela, l'opérateur d'inversion n'est pas très utilisé car l'opérateur de croisement, qui a la même fonction, est plus puissant vu qu'il peut combiner des allèles ou ensembles d'allèles de deux génotypes différents.

I.6.3 L'OPERATEUR DE MUTATION

L'opérateur de croisement peut donc générer toutes les combinaisons possibles d'allèles, même si la population ne contient qu'une seule copie de chaque allèle. Néanmoins, il arrive parfois que la dernière copie de certains allèles soit éliminée alors qu'elle aurait peut-être été utile dans des étapes ultérieures. Quand un allèle est perdu, l'opérateur de croisement n'a aucun moyen pour le réintroduire.

L'opérateur de mutation remplace aléatoirement un allèle d'un gène par un autre; de cette manière, il garantit qu'aucun allèle ne soit définitivement perdu de la population.

Mutation appliquée à $A = a_1...a_n$:

Pour i:=1 jusque n faire
< Remplacer l'allèle a_i par un autre allèle de ce gène
avec la probabilité P_M >

Figure 8 : L'opérateur de mutation

En général, la probabilité de mutation de chaque gène est très faible. L'opérateur de mutation est en fait un opérateur "d'arrière-plan" qui garantit à l'opérateur de croisement d'avoir un assortiment complet d'allèles.

L'opérateur de mutation introduit une diminution du nombre d'instances de schémas après l'étape de sélection. Toute instance d'un schéma ayant subi une mutation en une de ses positions définies n'est plus une instance de ce schéma. Toutefois, si la mutation a lieu en une autre position, elle restera une instance de ce schéma.

Variation
Mutation :
Pour k:=1 jusque M faire
< Appliquer l'opérateur de mutation à $A_k \in \beta'(t)$ >

Figure 9 : L'étape de variation avec l'opérateur de mutation

La probabilité pour un gène de subir une mutation est P_M. Si $\eta(\xi)$ est le nombre de positions définies d'un schéma, alors $(1-P_M)^{\eta(\xi)}$ est la probabilité qu'une instance de ξ reste une instance de ξ .

Les opérateurs de croisement et de mutation n'ont pas une grande incidence sur le nombre d'instances d'un schéma après l'étape de sélection.

I.7 DESCRIPTION COMPLETE D'UN ALGORITHME GENETIQUE

A présent, nous pouvons donner une description complète d'un AG (figure 10). On peut résumer sa façon de procéder comme suit :

L'étape de sélection disperse rapidement les bonnes combinaisons d'allèles, c'est-à-dire des instances de schémas ayant des caractéristiques supérieures

à la moyenne. Pendant l'étape de variation, l'opérateur de croisement analyse l'utilité de ces combinaisons d'allèles dans d'autres contextes; pour ce faire, il génère de nouvelles instances de schémas existants et simultanément des instances de nouveaux schémas. L'opérateur d'inversion regroupe des allèles qui s'accordent bien, et les rend ainsi moins sujet à la décomposition. L'opérateur de mutation a pour rôle de réintroduire des allèles perdus.

1. *Initialisation* : Mettre t à 0 et choisir au hasard
   M génotypes de α pour former $\beta(0)$.

2. *Sélection* : Pour k:=1 jusque M faire
   < Choisir un génotype $A_i(t)$ qui devient le
   k-ième génotype de $\beta'(t)$.
   Chaque $A_i \epsilon \beta(t)$ a une chance $\frac{\mu(A_i)}{\Sigma\mu(A_j)}$ d'être sélectionné >

3. *Variation* :
   Croisement : Pour k:=1 jusque (M div 2) faire
   < Choisir au hasard deux génotypes A_i et A_j dans $\beta'(t)$ >
   < Appliquer l'opérateur de croisement à A_i et A_j
   avec la probabilité P_c >
   Si < Le croisement est appliqué > alors
   <Remplacer A_i et A_j par les nouveaux génotypes>
   Inversion : Pour k:=1 jusque M faire
   < Choisir au hasard un génotype A_i dans $\beta'(t)$ >
   < Appliquer l'opérateur d'inversion à A_i avec la
   probabilité P_I >
   Si <L'opérateur d'inversion est appliqué> alors
   < Remplacer A_i par le nouveau génotype >
   Mutation :
   Pour k:=1 jusque M faire
   <Appliquer l'opérateur de mutation à $A_k \in \beta'(t)$>

4. Faire t=t+1 et retourner au point 2.

Figure 10 : Description complète d'un AG

L'AG décrit à la figure précédente peut être rendu beaucoup plus efficace. La figure 11 décrit un AG amélioré dans lequel les étapes de sélection et de variation sont plus ou moins entrelacées. Cet AG économise beaucoup de temps de calcul car il ne boucle qu'une seule fois dans la population et parce qu'il utilise moins de nombres aléatoires. Cependant, le comportement global des deux AG (figures 10-11) est le même.

1. *Initialisation* : Mettre t à 0 et choisir au hasard
   M génotypes dans α pour former $\beta(0)$.

2. *Sélection et variation* :
   Pour k:=1 jusque M faire
   < Choisir un génotype $A_i \in \beta(t)$ pour devenir le k-ième
   génotype de $\beta(t+1)$.
   Chaque $A_i \in \beta(t)$ a une chance $\frac{\mu(A_i)}{\Sigma\mu(A_i)}$ d'être
   sélectionné >
   Si < k est pair > alors
   < Croiser $A_k \in \beta(t+1)$ avec $A_{k-1} \in \beta(t+1)$ avec la
   probabilité P_c >
   Si <L'opérateur de croisement est appliqué> alors
   < Remplacer A_k et A_{k-1} par les nouveaux
   génotypes>
   < Appliquer l'opérateur d'inversion à $A_k \in \beta(t+1)$ avec
   la probabilité P_I >
   Si < L'inversion est appliquée > alors
   < Remplacer A_k par le nouveau génotype >
   < Appliquer l'opérateur de mutation à $A_k \in \beta(t+1)$ >

3. Faire t=t+1 et retourner au point 2.

Figure 11 : Un AG plus performant

Cet AG est appelé l'AG canonique; il est fondamentalement identique à celui proposé par Holland. La seule différence entre ces deux AG réside dans l'application de l'opérateur de croisement. Dans l'algorithme de Holland (figure 12), un des parents est choisi au hasard dans la population et il n'y a qu'un seul des enfants (choisi au hasard) qui remplacera le parent dans la population. Etant donné qu'un des parents n'est pas choisi en fonction de ses caractéristiques, les instances de schémas de caractéristiques supérieures à la moyenne augmenteront moins rapidement que dans le cas d'un AG canonique et les instances de schémas de caractéristiques inférieures à la moyenne diminueront moins rapidement. On ne peut toutefois pas affirmer que l'un de ces AG soit meilleur que l'autre. Pour certains problèmes de recherche, on peut aboutir avec cet AG à une meilleure solution, pour d'autres, de bonnes solutions peuvent ne pas être trouvées du tout. Le genre de problèmes de recherche traités le mieux par les AG reste à découvrir. Pour certains problèmes, l'une des méthodes sera plus appropriée que l'autre. Toutefois, à l'heure actuelle, on ne sait pas encore lequel de ces deux AG convient le mieux pour un type de problème donné.

1. *Initialisation* : Mettre t à 0 et choisir au hasard
   M génotypes dans α pour former $\beta(0)$.
2. *Sélection et variation* : Pour k:=1 jusque M faire
   < Choisir un génotype $A_i \in \beta(t)$ pour devenir le k-ième
   génotype de $\beta(t+1)$. Chaque $A_i \in \beta(t)$ a une chance $\frac{\mu(A_i)}{\Sigma\mu(A_j)}$ d'être sélectionné >
   < Choisir $A_i \in \beta(t)$ avec une probabilité uniforme >
   < Croiser $A_i \in \beta(t)$ avec $A_k \in \beta(t+1)$ avec la probabilité P_C >
   Si <L'opérateur de croisement est appliqué> alors
   < Choisir au hasard un des génotypes résultants >
   < Remplacer $A_k \in \beta(t+1)$ par ce génotype >
   < Appliquer l'opérateur d'inversion à $A_k \in \beta(t+1)$ avec
   la probabilité P_I >
   Si < L'inversion est appliquée > alors
   < Remplacer A_k par le nouveau génotype >
   < Appliquer l'opérateur de mutation à $A_k \in \beta(t+1)$ >
3. Faire t=t+1 et retourner au point 2.

Figure 12 : L'AG proposé par HOLLAND

Les AG ne se limitent pas à ceux décrits ci-dessus; il en existe bien d'autres. Dans tous les travaux réalisés par HOLLAND, les chromosomes sont des chaînes binaires - des listes de 0 et de 1. Plusieurs chercheurs ont étudié l'utilisation d'autres représentations [7]. En fait, chaque chercheur semble utiliser une version différente de l'algorithme. Ces différentes versions portent également le nom d'AG parce qu'elles ont toutes les caractéristiques suivantes [5] :

- Chaque point de l'espace de recherche est considéré comme un génotype (c'est-à-dire comme un ensemble de variables de taille fixe).

- Une fonction d'évaluation qui joue le rôle de l'environnement en estimant chaque génotype (c'est-à-dire chaque solution) selon son caractère.

- Une population (de taille fixe) de génotypes, initialisée au hasard.

- Un processus de sélection, reproduisant des génotypes en fonction de leur caractère.

- Des opérateurs génétiques pour modifier la compositon de la progéniture durant la reproduction.

- Des valeurs pour les paramètres utilisés par l'AG (taille de la population, probabilité d'appliquer les opérateurs génétiques)

Notons que l'utilisation des opérateurs génétiques ne se limite pas aux opérateurs de croisement, d'inversion et de mutation.

CHAPITRE II : APPLICATIONS DES ALGORITHMES GENETIQUES

II.1 INTRODUCTION

Dans ce chapitre, nous nous intéresserons à quelques problèmes qui ont été résolus par les AG.

En pratique, un AG doit avoir les éléments suivants pour résoudre un problème :

- des génotypes qui représentent les solutions du problème
- une fonction "caractère" évaluant ces solutions
- des opérateurs génétiques
- des valeurs pour les paramètres utilisés par l'AG (par exemple la taille de la population et les probabilités d'appliquer les opérateurs génétiques)

Les deux premiers éléments dépendent de l'application. Les opérateurs génétiques les plus utilisés sont le croisement, l'inversion et la mutation qui doivent parfois être modifiés pour convenir à un problème particulier.

II.2 OPTIMISATION DE PARAMETRES

On rencontre les problèmes d'optimisation de paramètres dans divers domaines tels que la conception de bons systèmes d'exploitation, de raffineries de pétrole, d'auditoires avec une bonne acoustique et d'avions bien profilés. Dans chaque cas, il s'agit de trouver les valeurs des paramètres qui rendent les performances minimales ou maximales.

La mesure de performances de systèmes complexes est une fonction extrêmement complexe à plusieurs dimensions, ce qui exclut l'utilisation de techniques standards d'optimisation. C'est pourquoi on fait généralement appel à des techniques telles que les AG.

Soit $x_1, x_2, ..., x_N$, les N paramètres de contrôle d'un processus donné. Dans toutes les situations pratiques, les x_i seront soumis à des contraintes internes de la forme $a_i \leq x_i \leq b_i$. Les génotypes de l'AG représenteront les paramètres de contrôle et le caractère d'un génotype, la performance d'un processus complexe avec ces paramètres de contrôle. Chaque x_i sera représenté par une chaîne binaire. Donc, les valeurs de x_i comprises dans l'intervalle $[a_i,b_i]$ sont représentées par

$$L_i = \log_2 [(b_i-a_i)\, \Delta x_i]$$

bits, où Δx_i représente le niveau de résolution souhaité.

Chaque génotype est donc une chaîne binaire de longueur

$$L = \sum_{i=1}^{N} L_i$$

Prenons un exemple simple pour montrer comment un AG parcourt rapidement et efficacement l'espace des solutions. Soit un paramètre x qu'il faut optimiser dans l'intervalle [0,1] avec une précision de un pour un million. Chaque génotype est donc une chaîne binaire de longueur

$$\log_2[(1-0)10^6] = 20 \text{ bits}$$

Supposons que la mesure de performances à maximiser soit de la forme suivante :

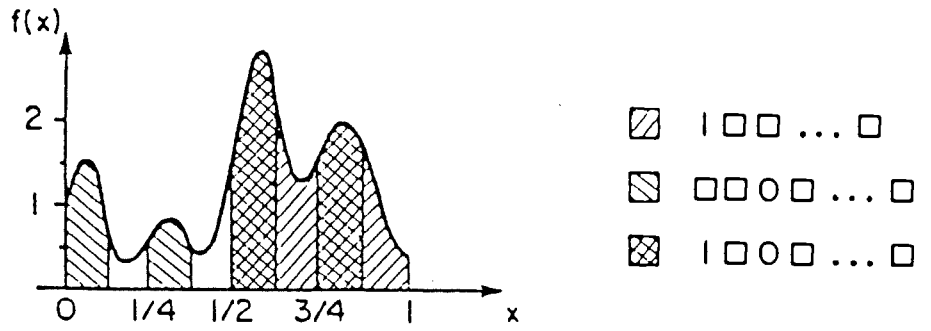


Figure 13 : Fonction simple à maximiser

Cette figure nous donne une interprétation concrète des schémas. Le schéma 1[]...[] est la moitié de droite de la surface [1/2,1], tandis que le schéma [][]0[]...[] est un ensemble de quatre bandes et le schéma 1[]0[]...[] est l'intersection des deux schémas précédents. Il est intéressant de comparer deux représentations différentes du paramètre de contrôle x. Six allèles de dix valeurs possibles peuvent produire sensiblement la même quantité de génotypes différents que vingt allèles avec deux valeurs possibles, puisque $10^6 \simeq 2^{20}$. Cependant, le nombre de schémas est très différent dans ces deux cas : $11^6 \simeq 1.7\ 10^6$ contre $3^{20} \simeq 3.4\ 10^9$. De plus, dans le premier cas, chaque $A \in \alpha$ n'est une instance que de $2^6 = 64$ schémas, alors que dans le second cas, chaque $A \in \alpha$ est une instance de $2^{20} = 10^6$ schémas. C'est pourquoi, pour les AG, il est préférable d'avoir beaucoup de gènes et peu d'allèles par gène, ce qui est habituellement le cas en génétique, plutôt que peu de gènes et beaucoup d'allèles par gène.

A la figure 13, on peut voir que la valeur moyenne de tous les points dans le schéma 1[]...[] est approximativement 1.5, pour le schéma [][]0[]...[], la moyenne est approximativement 1 et pour le schéma 1[]0[]...[], approximativement 2. Dès lors, les instances de 1[]...[] s'accumuleront plus rapidement que celles de [][]0[]...[], et les instances de 1[]0[]...[] s'accumuleront encore plus rapidement. C'est de cette manière qu'un AG localise rapidement l'optimum global d'une fonction.

Illustrons ceci en exécutant un cycle d'un AG qui n'utilise que l'opérateur de croisement. Supposons que l'on commence avec une population de cinq génotypes (dans les applications courantes, la taille est de l'ordre de 100). La table suivante reprend les cinq génotypes et leur probabilité d'être choisis dans l'étape de sélection. Nous n'avons utilisé que six bits.

A_i	$\mu(A_i)$	$\mu(A_i)/\Sigma\mu(A_j)$
.001100	1/2	2/25
.000100	3/2	6/25
.101000	2	8/25
.110011	7/4	7/25
.011100	1/2	2/25

La figure 14 décrit la position des génotypes. On peut voir que c'est A_3 qui aura le plus de chances d'être reproduit.

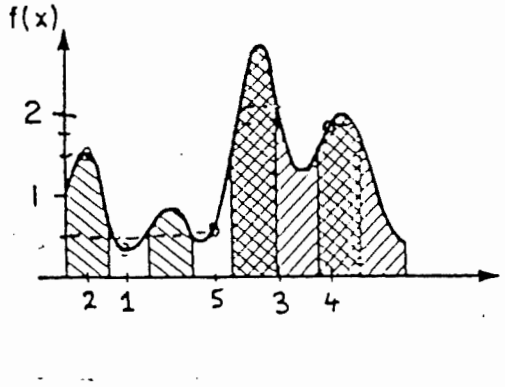


Figure 14 : Position des génotypes initiaux

A la figure 15, on trouve le résultat d'un AG qui n'utilise que l'opérateur de croisement.

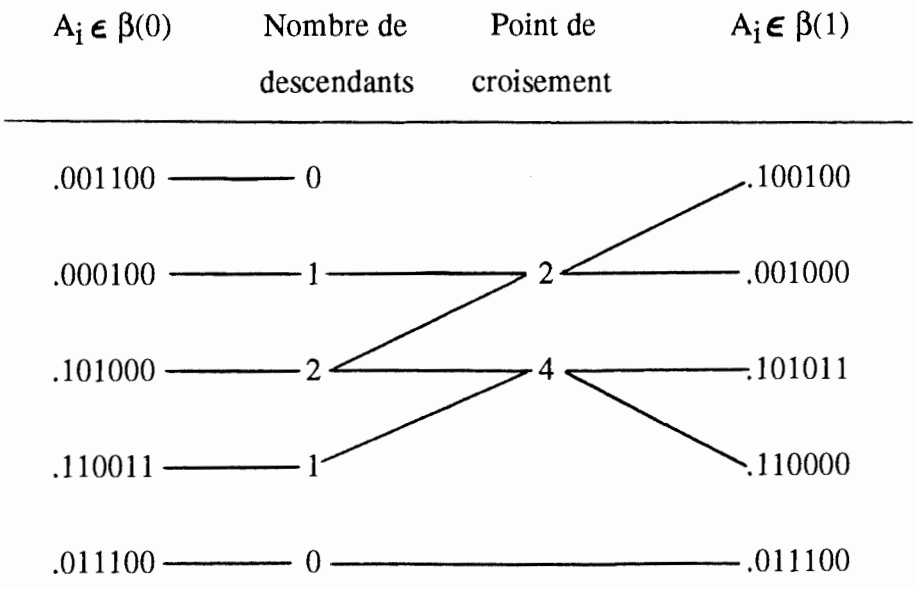


Figure 15 : Résultat d'un AG n'utilisant que l'opérateur de croisement

Les génotypes obtenus sont représentés à la figure 16. On remarque qu'il y a à présent deux instances du schéma le plus prometteur.

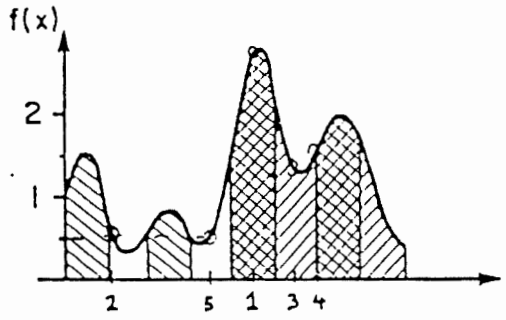


Figure 16 : Emplacement des génotypes après un cycle d'AG

L'exemple précédent est évidemment très limité; toutefois, il apparaît clairement qu'au fil des cycles, les génotypes auront de plus en plus de chances de se grouper dans les régions les plus prometteuses de la fonction.

II.3 LE PROBLEME DU VOYAGEUR DE COMMERCE

Le problème du voyageur de commerce (PVC) est un problème célèbre parce qu'il est facile à énoncer mais difficile à résoudre. En prenant l'avion, un voyageur de commerce doit visiter successivement un certain nombre de villes; il doit passer une fois seulement par chaque ville, au cours d'un circuit qu'il veut de longueur minimale et qui doit le faire revenir au point de départ. Une stratégie consiste à chercher tous les circuits possibles et à retenir le plus court, mais même pour un problème d'énoncé aussi simple, cette méthode d'énumération est impraticable : avec seulement 12 villes à visiter, le nombre de circuits possibles est déjà égal à 19 959 400. Bien entendu, nous pouvons nous dispenser d'envisager les circuits manifestement trop longs, par exemple ceux qui présentent de nombreux croisements, mais le nombre de circuits reste trop important, et la solution exacte est hors d'atteinte des gros ordinateurs dès que la "taille" du

problème, en l'occurence le nombre de villes est tant soit peu importante. La difficulté et l'enjeu économique de tels problèmes a incité les mathématiciens à étudier des méthodes de résolution plus efficaces, sur ordinateur. Le PVC a servi de problème d'essai pour beaucoup d'algorithmes de recherche. C'est pourquoi nous allons essayer de le résoudre au moyen d'un AG dans un premier temps et ultérieurement, par la méthode du recuit simulé. Les génotypes de l'AG représenteront les différents circuits possibles et le caractère d'un génotype sera la distance totale associée au circuit. Les génotypes possédant de petites valeurs de caractère seront favorisés dans l'étape de sélection.

Malheureusement, le choix d'une représentation appropriée pour les circuits convenant aux opérateurs génétiques est une tâche fastidieuse. Supposons que le circuit soit représenté par une liste de villes. Si nous appliquons l'opérateur de croisement classique aux génotypes abcde (chaque lettre représentant une ville) et adecb en x=3, nous obtenons les circuits abccb et adede qui ne sont pas acceptables (on passe deux fois par deux mêmes villes).

Ceci est un problème fréquemment rencontré lors de l'utilisation des AG. En fait, le PVC est un problème d'optimisation de paramètres où chaque gène peut considérer chaque ville comme un allèle mais deux gènes ne peuvent pas avoir le même allèle.

En général, on peut imposer certaines contraintes aux génotypes en attribuant des pénalisations à ceux qui ne satisfont pas à ces contraintes ou en créant des opérateurs génétiques qui évitent la formation de génotypes ne respectant pas les contraintes. Chacune de ces possibilités a ses avantages et ses inconvénients.

Si un génotype est fortement pénalisé suite à la violation de contraintes et si l'application a beaucoup de chances de générer des génotypes violant les contraintes, on risque de créer un AG qui consacre la plupart de son temps aux génotypes illégaux. En outre, quand un génotype légal est trouvé, il risque de chasser les autres et dans ce cas, la population convergera vers ce dernier sans trouver de meilleurs génotypes; en effet, les chemins susceptibles de conduire à d'autres génotypes légaux nécessitent la

production de génotypes illégaux et ces derniers sont incapables de se reproduire car ils sont pénalisés pour violation de contraintes.

Si on impose des pénalités modérées, le système peut développer des génotypes qui violent les contraintes mais qui seront mieux appréciées que ceux qui ne le font pas vu que la fonction "caractère" peut être mieux atteinte en acceptant des pénalités modérées.

Dans le cas du PVC, la plupart des chercheurs ont implémenté des AG qui évitent la construction de génotypes illégaux. Pour l'implémentation présentée ici, nous avons modifié l'opérateur de croisement comme suit :

Dans le premier circuit (génotype), un sous-circuit est choisi au hasard. Le second circuit est ensuite analysé pour déterminer si il contient un sous-circuit qui visiterait les mêmes villes que le premier sous-circuit (cette partie est évidemment du traitement intensif). Par exemple, supposons que le premier circuit soit abcdefg, le second gcadbfe et le sous-circuit choisi dans le premier circuit abcd. Le second circuit contient le sous-circuit cadb qui visite les mêmes villes que le premier sous-circuit. Le croisement classique est applicable à ces sous-circuits, produisant deux nouveaux circuits cadbefg et gabcdfe. Si aucune permutation du premier sous-circuit n'est trouvée dans le second génotype, l'opérateur de croisement n'est pas appliqué. La probabilité d'appliquer l'opérateur de croisement (P_C) peut être plus ou moins ajustée en choisissant plusieurs sous-circuits dans le premier génotype et en recherchant des permutations de chacun d'entre eux dans le second génotype. Nous avons trouvé qu'il est suffisant de laisser l'opérateur de croisement choisir un seul sous-circuit. Dans ce cas, la probabilité d'appliquer l'opérateur de croisement est d'environ 20 % (P_C = 20 %).

Par contre, l'opérateur d'inversion peut être utilisé sans aucune modification; toute inversion d'un sous-circuit produira un nouveau circuit légal. De plus, la distance totale d'un nouveau circuit peut être calculée très facilement car il n'y a que les distances aux extrémités du sous-circuit qui doivent être recalculées. L'opérateur d'inversion est appliqué à environ 40 % de la population (P_I = 40%). Les opérateurs de croisement et d'inversion formeront une excellente paire car l'inversion construit de bons sous-circuits et le croisement les disperse dans la population. L'opérateur de mutation

n'est pas utilisé car l'opérateur d'inversion et l'opérateur de croisement modifié rendent impossible le fait qu'un gène perde un allèle.

La figure 17 décrit un PVC à 64 villes résolu par un AG utilisant une population de 50 génotypes et les opérateurs génétiques décrits ci-dessus. Cette solution fut obtenue après 600 générations.

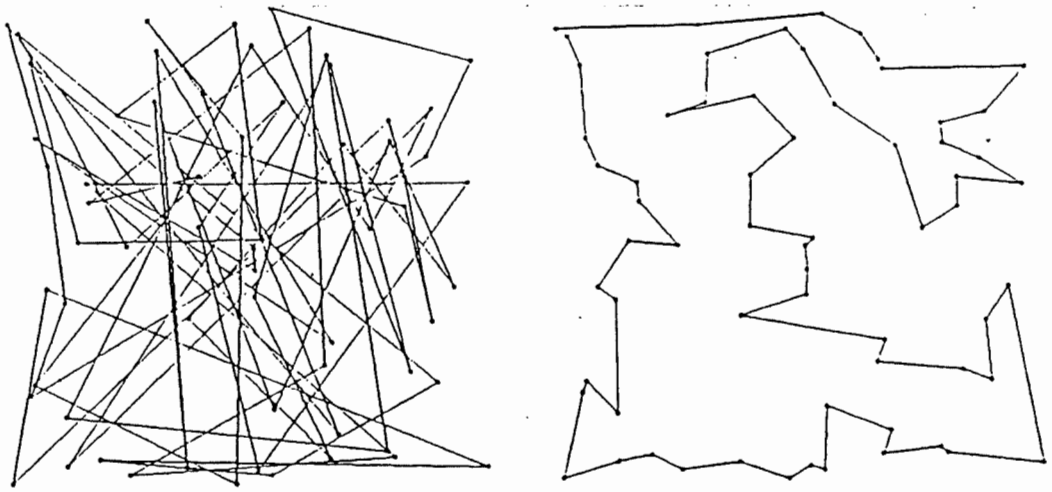


Figure 17 : Circuit initial arbitraire et solution d'un PVC à 64 villes

La figure 18 nous montre l'évolution de la longueur d'un circuit de 64 villes obtenu par un AG sur 600 générations.

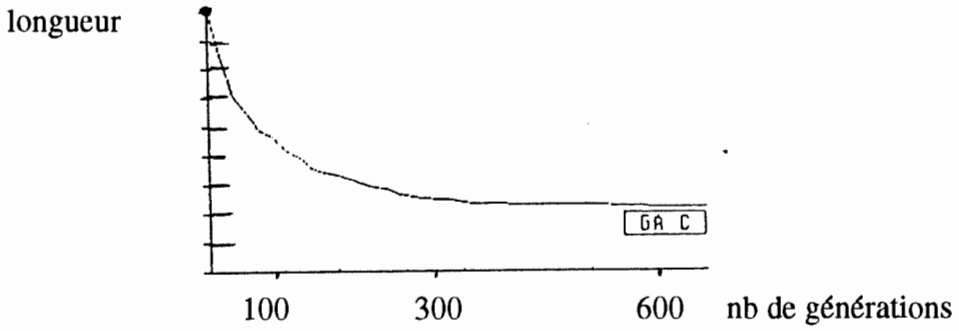


Figure 18 : L'évolution complète de l'AG C

Cet AG utilise l'opérateur de croisement modifié en plus de l'opérateur d'inversion. Nous pouvons voir qu'il y a une évolution rapide jusqu'à environ 450 générations et qu'ensuite, la courbe se stabilise. La population a convergé vers une solution et il n'y a plus d'amélioration possible. Notons toutefois que chaque exécution de l'algorithme donne des solutions

différentes au PVC à 64 villes, mais toutes ces solutions ne diffèrent pas de plus de 2%.

Il est difficile de déterminer le niveau de la qualité des solutions obtenues car la longueur exacte minimale du circuit est inconnue. Pour un PVC avec N villes, il y a (N-1)!/2 circuits possibles. Pour un PVC à 64 villes, nous avons donc 63!/2 = 10^{87} circuits possibles. Puisque la seule manière de trouver la longueur exacte du circuit le plus court est de considérer presque chaque circuit, il est impossible de trouver cette longueur minimale en un temps raisonnable.

CHAPITRE III : AMELIORATION DES PERFORMANCES DES AG

III.1 INTRODUCTION

Il reste encore beaucoup à apprendre sur la manière d'implémenter effectivement un AG. Bien que les performances de la plupart des implémentations soient comparables ou meilleures que les performances d'autres techniques de recherche, on ne se montre pas toujours à la hauteur des espérances révélées par la théorie. Ceci se manifeste en pratique par une convergence de la recherche vers une solution sous-optimale.

Un AG convergera vers une solution sous-optimale lorsque les génotypes de la population sont identiques ou presque. Dans ce cas, l'opérateur de croisement cesse de produire de nouveaux génotypes et l'algorithme réalise tous ses essais dans une région restreinte de l'espace. Malheureusement, ce phénomène appelé "convergence prématurée" se produit habituellement avant d'atteindre le véritable optimum.

Pour éviter cette convergence prématurée, de nombreuses solutions ont été proposées [8].L'importance de ce phénomène fut mis en évidence pour la première fois par Dejong. Celui-ci démontra que limiter l'erreur dans les probabilités de sélection pour chaque chaîne améliore substantiellement la performance de l'algorithme. D'autres tels que Grefenstette ont essayé de réduire l'erreur en optimisant les paramètres de l'algorithme. Nous exposerons ultérieurement le plus efficace de ces remèdes. La relation entre performances et convergence d'un AG est la suivante : si un AG converge prématurément, les performances seront mauvaises, de même que si un AG converge très lentement. Les meilleures performances sont obtenues quand l'algorithme converge au bon moment.

III.2 DETERMINATION DES PARAMETRES DE CONTROLE

La valeur des paramètres de contrôle (par exemple la taille de la population, les probabilités d'appliquer les opérateurs génétiques) est primordiale pour éviter la convergence prématurée [4].

1) La taille de la population (M)

Les AG ne donnent pas les performances escomptées quand la taille de la population est faible car les données sont alors insuffisantes pour la plupart des schémas. Une grande population sera plus susceptible de contenir les instances d'un grand nombre de schémas. L'AG pourra donc entreprendre une recherche plus poussée. En conséquence, une grande population décourage le phénomène de convergence prématurée vers une solution sous-optimale. Toutefois, une grande population nécessite plus d'évaluations de fonction par génération, ce qui pourrait se traduire par une vitesse de convergence trop lente pour être acceptable.

2) Le taux de croisement (P_C)

Le taux de croisement contrôle la fréquence d'application de l'opérateur de croisement. Dans chaque nouvelle population, P_C*M structures subissent le croisement. Les nouveaux génotypes sont introduits dans la population d'autant plus rapidement que le taux de croisement est élevé. Si le taux de croisement est trop faible, la recherche risque de ne pas évoluer à cause du faible taux d'exploration. Par contre, si le taux de croisement est trop élévé, les meillleures structures sont écartées avant que l'étape de sélection ne les prenne en compte. Habituellement, on réalise les expériences avec des taux de croisement variant de 0.25 à 1.00 par incréments de 0.05.

3) Le taux de mutation (P_M)

La mutation est un opérateur secondaire qui accroît la diversité dans la population. Après la sélection, chaque élément de chaque structure de la nouvelle population subit un changement aléatoire avec une probabilité égale au taux de mutation P_M. Par conséquent, P_M*M*N

mutations se produisent à chaque génération, N étant la longueur de chaque structure. Un faible niveau de mutation permet d'éviter qu'un gène ne converge continuellement vers une seule valeur. Un taux de mutation plus élevé empêche la convergence prématurée. Toutefois, si le taux de mutation est trop grand, la recherche devient essentiellement une recherche aléatoire. Les expériences courantes autorisent huit valeurs pour le taux de mutation, augmentant exponentiellement de 0.0 à 1.0.

4) L'intervalle de génération (G)

L'intervalle de génération contrôle le pourcentage de la population à remplacer durant chaque génération. Il y a M*(1 - G) structures de la population P(t) qui sont choisies au hasard pour demeurer intactes dans la population P(t + 1). Une valeur de G = 1.0 signifie que la population toute entière est remplacée à chaque génération. Une valeur de G = 0.5 signifie que la moitié des structures de chaque population subsistent à la génération suivante. Dans les expériences courantes, on fait varier G de 0.30 à 1.00, par incréments de 0.10.

5) La stratégie de sélection (S)

Deux stratégies différentes peuvent être mises en oeuvre lors des expériences :

- la sélection pure (S=P) pour laquelle chaque structure de la population courante a d'autant plus de chances de se reproduire que ses performances sont élevées.
- la stratégie élitiste (S=E) consiste à garder intacte la meilleure structure pour la génération suivante. En l'absence d'une telle stratégie, les meilleures structures risquent de disparaître en raison d'une erreur d'échantillonnage, d'un croisement ou d'une mutation.

Bien que l'on sache comment les variations d'un paramètre influencent les performances d'un AG, on peut très difficilement prévoir comment les différents paramètres vont interagir. Par exemple, quel sera l'effet d'une augmentation de la taille de la population quand on diminue le taux de croisement ?

Les expériences réalisées par Grefenstette [4] ont montré que les valeurs "standards" pour les paramètres c'est-à-dire M=50, P_C=0.6 et P_M=0.001 donnent de bons résultats pour de nombreux problèmes. Des taux de mutation supérieurs à 0.05 donnent en général de mauvais résultats. En fait, chaque paramètre possède une plage de valeurs pour laquelle les résultats sont meilleurs. Pour la taille de la population, par exemple, les meilleurs résultats sont obtenus quand on a entre 30 et 100 génotypes (figure 19).

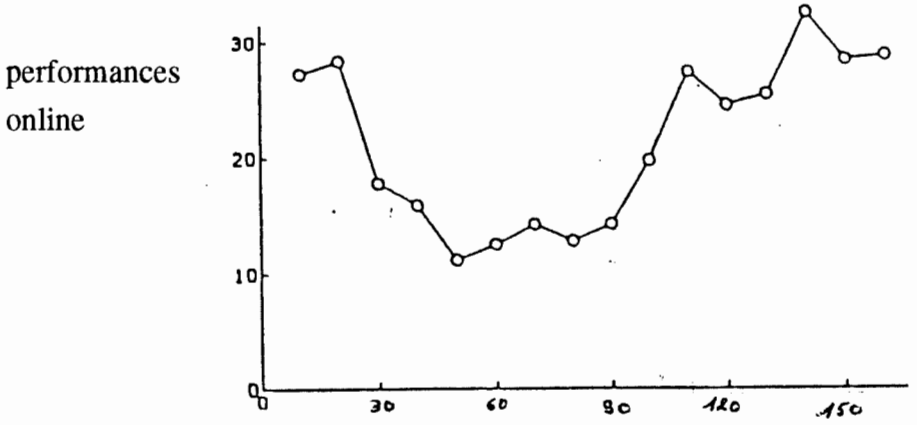


Figure 19 : Performances moyennes d'un AG en fonction de la taille de la population

Lorsqu'on désire optimiser une fonction numérique f(x) à l'aide d'un AG, on définit en général la valeur de la performance u(x) d'une structure x par $u(x)=f(x)-f_{min}$, où f_{min} représente la valeur minimale que f(x) est supposée prendre dans l'espace de recherche donné. La valeur u(x) est toujours positive quel que soit le type de la fonction f(x). En général, f_{min} n'est pas connue à priori puisque c'est l'objet de la recherche. Dans ce cas, il est plus approprié de définir $u(x)=f(x)-f(x_{min})$ où $f(x_{min})$ est la valeur minimale de toutes les structures évaluées précédemment. Pour. les petites populations (de 20 à 40 génotypes), on obtient les meilleurs résultats lorqu'on a simultanément un taux de croisement élevé et un faible taux de mutation ou un faible taux de croisement et un taux de mutation élevé puisque le faible nombre d'éléments différents est compensé par un grand taux de croisement ou de mutation. Pour des populations de taille moyenne (de 30 à 90 génotypes), le taux de croisement optimal diminue quand la taille de la population augmente puisque dans les plus petites populations, les génotypes les plus performants ont plus de chances de dominer rapidement la population. Dans ce cas, un taux de croisement plus élevé permet d'éviter le phénomène de convergence prématurée.

III.3 MESURES DES PERFORMANCES D'UN AG

On considère deux mesures de performances pour des stratégies de recherche à savoir les performances *online* et les performances *offline* [4].

La performance online d'une stratégie de recherche s pour une surface de réponse e est définie comme suit :

$$U_e(s, T) = moy_t\,(u_e\,(t)) = 1/T * \sum_{t=1}^{T} u_e\,(t) \qquad t = 0,1, ..., T$$

où u_e (t) est la performance moyenne des structures évalulées à l'instant t et T le nombre total de générations. La performance *online* est donc la moyenne des performances de toutes les structures considérées au cours de la recherche.

La performance *offline* d'une stratégie de recherche s pour une surface de réponse e est définie comme suit :

$$U_e^*(s,T) = moy_t(u_e^*(t)) = 1/T * \sum_{t=1}^{T} u_e(t) \qquad t = 0,1, ...,T$$

où $u_e^*(t)$ est la meilleure performance obtenue dans l'intervalle de temps [0,t].

III.4 LA DIVERSITE DANS LA POPULATION

L'optimisation des paramètres de contrôle ne permet pas toutefois d'éviter le phénomène de convergence prématurée. C'est pourquoi nous nous intéresserons à d'autres stratégies qui devraient permettre de résoudre ce problème. Mauldin [8] a proposé une solution radicale qui, malheureusement, ne respecte pas les principes de la théorie de la génétique. Il commence par mettre en évidence que la convergence prématurée est due au fait que les génotypes de la population se ressemblent beaucoup trop. Pour résoudre le problème de la convergence prématurée, il faudra donc garantir que tous les membres de la population

restent bien différents. On impose la diversité dans la population en définissant une valeur unique comme la distance minimale de Hamming (c'est-à-dire le nombre de bits qui ne sont pas semblables) acceptable entre tous les descendants et tous les génotypes existant dans la population. A chaque fois qu'un génotype ne répond pas à cette condition, les valeurs d'allèles semblables sont aléatoirement modifiées jusqu'à ce que la distance minimale de Hamming [annexe 2] soit respectée. Pour garantir la convergence de l'AG, la valeur unique de k bits est décrémentée lentement jusqu'à un bit pendant la recherche en utilisant la relation :

$$\text{distance de Hamming} \geq k_t = k\,(N-t)/N$$

où N est le nombre total de générations et k_t la valeur unique à la génération t. Donc, au début de la recherche, l'espace est échantillonné grossièrement selon une grille et au cours de la recherche la taille de la grille diminue progressivement. Ce processus ressemble au recuit simulé, la distance minimale pouvant être assimilée à la température qui décroît (voir chapitre IV).

Mauldin [8] a démontré que cette méthode améliore le taux de convergence de la recherche vers une bonne solution. Toutefois, cette méthode présente deux inconvénients. D'une part, si les génotypes sont longs ou si la population est grande, l'implémentation de cette méthode coûte cher. D'autre part, le fait de forcer les AG à échantillonner l'espace de cette manière empêche les chaînes parents de transmettre de bons schémas à leurs descendants. En manipulant la diversité dans une population, on ne fait que supprimer les premiers symptômes du problème de convergence prématurée, sans pour autant en éliminer les causes.

III.5 LA CONVERGENCE PREMATUREE

Une façon de prévoir la convergence prématurée dans les AG consiste à chercher les indicateurs qui témoignent d'un affaiblissement de la diversité. On constate que la convergence prématurée se produit souvent après qu'un génotype ou un petit groupe de génotypes ait eu beaucoup de descendants.

La taille de la population étant limitée, si un génotype a un grand nombre de descendants, le reste de la population ne pourra avoir que peu de descendants. De même, si trop de génotypes n'ont aucun descendant, il en résulte une perte rapide de la diversité qui entraîne le phénomène de convergence prématurée. En tenant compte du pourcentage de la population produisant des descendants (le taux de participation), on peut prévoir cette convergence prématurée et avoir une chance de l'éviter.

III.6 LE TAUX DE PARTICIPATION

Le taux de participation, pourcentage de la population courante produisant des descendants, est un bon indicateur pour prévoir la convergence prématurée. La meilleure méthode pour éviter ce phénomène consiste à éviter que le taux de participation soit faible. Dans les AG, l'exploration est principalement réalisée par l'opérateur de croisement. Dès lors, on peut faire l'hypothèse que des améliorations dans la manière d'implémenter le croisement aideront à éviter la convergence prématurée associée à un faible taux de participation.

La première modification que l'on peut apporter à l'opérateur de croisement concerne le nombre de points de croisement. On sélectionne deux points de croisement au lieu d'un seul et on échange le segment situé entre ces deux points. L'avantage réside dans le fait que de longs schémas auront moins de chances d'être perturbés par l'opérateur de croisement.

Une seconde modification de l'opérateur de croisement consiste à limiter la position des points de croisement. Il arrive fréquemment que deux génotypes soient différents mais que les points de croisement tombent de telle façon que l'on doive échanger un segment identique. Ceci conduit à deux génotypes identiques aux originaux. Ceci s'oppose à la génération d'instances de nouveaux schémas.

En définitive, les modifications à apporter à l'opérateur de croisement sont simples : si on a deux génotypes, on prendra leurs représentants réduits contenant seulement les allèles qui diffèrent, on choisira au moins un des

points de croisement parmi ces représentants et on appliquera le croisement aux génotypes initiaux. Par exemple, les représentants réduits pour les génotypes 010110100 et 001000100 sont respectivement 1011 et 0100. Un des points de croisement doit donc être choisi entre les positions 1 et 4 [8].

III.7 UNE NOUVELLE METHODE

La majorité des modifications proposées jusqu'à présent donnent lieu à des versions artificielles d'AG. Voyons maintenant d'autres modifications des AG dites naturelles car elles sont basées sur des processus trouvés dans la nature.

III.7.1 LA DERIVE GENETIQUE

Dans la théorie des AG, si la quantité de bons schémas augmente dans la population, le nombre de schémas moins désirables diminuera puisque la taille de la population doit rester constante. La recherche converge vers une solution car les génotypes qui sont des instances de très bons schémas apparaissent en grand nombre. Un génotype parent ne peut produire qu'un nombre entier de descendants; par conséquent, le nombre d'instances d'un schéma ne peut refléter la proportion désirée avec une précision arbitraire dans une population finie. Ceci est une source d'erreur qui, associée aux processus de sélection et de variation peut engendrer de grandes dérives dans le nombre d'instances d'un schéma par rapport aux prévisions théoriques. Au cours des itérations de l'algorithme, l'effet de telles erreurs va se cumuler à un point tel que des instances de schémas utiles peuvent disparaître de la population. Ce phénomène, longtemps suspecté comme la cause principale de la convergence prématurée dans les AG, est appelé *dérive génétique*. La biologie s'est aussi longuement penchée sur ce problème.

Des exemples de changements génétiques dus à la dérive génétique ont été signalés chez l'homme. Dans des petites populations isolées depuis longtemps pour des raisons géographiques (eskimos), religieuses (petites

sectes), politiques ou sociales, la dérive génétique s'est notamment manifestée par une perte complète de certains allèles responsables du groupe sanguin (le groupe sanguin d'un individu est déterminé par trois allèles : I^A, I^B et I^O). La dérive génétique dépend notamment de la taille de la population et du taux de mutation. Plus la population est grande ou plus le taux de mutation est élevé, plus la dérive génétique sera petite.

En AG, nous voulons réduire la dérive génétique pour améliorer les performances. Cependant, nous avons vu précédemment que si on augmente la taille de la population ou le taux de mutation, les performances des AG diminuent. Il est dès lors impératif de chercher d'autres stratégies pour réduire la dérive génétique dans les AG.

Une méthode simple consiste à laisser un certain nombre de populations se développer simultanément et indépendamment l'une de l'autre. On simule ainsi l'isolement géographique de sous-populations de la même espèce. Par la suite, nous utiliserons le terme *cellule* pour désigner les populations locales. Dans la nature, l'isolement géographique est un événement très important. Il est généralement considéré comme la première étape dans la formation d'une nouvelle espèce. La probabilité que la dérive génétique soit nuisible à l'évolution de toutes les cellules diminue si le nombre de cellules augmente. Illustrons ceci par un exemple : soit la probabilité F ($0<F<1$) que l'évolution d'une cellule s'arrête à cause de la dérive génétique. Si on a n cellules, la probabilité que l'évolution de toutes les cellules s'arrête est plus petite : F^n.

Dans la nature, les cellules sont rarement des systèmes fermés; il est toujours possible que des membres d'une cellule traversent la barrière et deviennent membre d'une autre cellule. Ce processus est appelé *échange de gènes* ou, si c'est un processus répétitif, *flux de gènes*. Le *flux de gènes* est un événement rare mais quand il se produit, il est très significatif : il retarde la formation de l'espèce mais atténue les effets de la dérive génétique. La notion de flux de gènes peut très facilement être intégrée dans un AG.

Le flux de gènes est une source de variation supplémentaire pour chaque cellule. Il s'agit d'un processus semblable à celui de la mutation car il est capable de réintroduire des allèles perdus, mais il

ressemble plus à l'opérateur de croisement du point de vue puissance. Il est très probable que quelques cellules contiennent des instances de différents schémas utiles. Le flux de gènes fait en sorte que ces instances deviennent des éléments de la même cellule. Ainsi, une recherche beaucoup plus poussée est possible. Ceci est fondamental pour éviter un faible taux de participation et donc empêcher la convergence prématurée. Il est toutefois indispensable que le flux de gènes soit un événement rare sinon le résultat obtenu serait le même qu'en travaillant avec une seule grande population.

III.7.2 UN NOUVEL ALGORITHME

Il existe plusieurs façons d'intégrer dans un AG la notion de flux de gènes entre cellules. Pour garder l'analogie avec l'isolement géographique dans la nature, les n cellules seront placées dans une grille à deux dimensions. Le flux de gènes ne se produira qu'entre des cellules adjacentes [3].

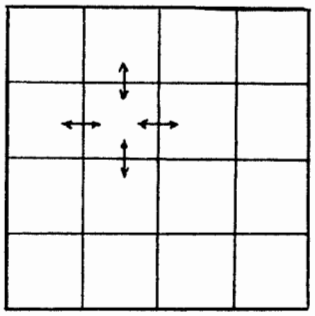


Figure 20 : Une grille 4x4 avec 16 cellules

Le taux de mutation supplémentaire doit être faible puisque le travail relatif à l'opérateur de mutation est en partie exécuté par le processus flux de gènes.

Un paramètre supplémentaire, appelé *taux de migration* (m) doit être introduit pour contrôler le flux de gènes. Le taux de migration représente le nombre de génotypes échangés entre une cellule et sa

voisine après chaque génération. Par exemple, si m=5, cela signifie qu'il y aura 5 génotypes qui seront échangés lors de chaque génération et si m=0.2, il n'y aura qu'un seul génotype échangé toutes les 5 générations.

Les génotypes qui sont échangés, c'est-à-dire ceux qui émigrent vers une cellule adjacente, sont choisis en fonction de leurs caractères, tout comme dans l'étape de sélection d'un AG. Pour chaque cellule voisine, le processus de sélection est répété. Si une cellule transmet un certain nombre de ses meilleurs génotypes à ses voisins, cela portera sans doute préjudice à la recherche dans cette cellule. Par conséquent, seules des copies des génotypes sont envoyées aux cellules adjacentes. Etant donné que la taille doit rester constante, les génotypes les moins bons seront retirés de la population.

Les quatre points ci-dessous nous donnent une esquisse d'un AG incorporant la notion de flux de gènes.

Pour chaque cellule :

1. *Initialiser* la population au hasard.

2. *Exécuter* un AG "standard" pour [si m<1 alors m^{-1} sinon 1] générations.

3. *Choisir* [si m > 1 alors m * le nombre de voisins sinon le nombre de voisins] génotypes selon leurs caractères et *envoyer* des copies de ceux-ci aux cellules adjacentes (m à chacune ou une à chacune si m<1).

4. *Recevoir* [si m > 1 alors m*le nombre de voisins sinon le nombre de de voisins] génotypes et *remplacer* les génotypes les moins bons par ces génotypes.

III.7.3 RESULTATS EXPERIMENTAUX [3]

Comparons à présent les performances des algorithmes en considérant les résultats expérimentaux obtenus sur 5000 évaluations de fonctions.

Une première expérience a été réalisée en considérant des cellules qui évoluent simultanément sans flux de gènes (m=0). Dans ce cas, chaque cellule a 50 génotypes et les opérateurs génétiques sont appliqués avec les probabilités P_C =0.6 et P_M=0.001 à chaque génotype de toutes les cellules. La figure 22 montre les résultats de cette expérience. D_n signifie que n cellules se développent; dès lors, D_1 représente la courbe d'un AG traditionnel. Il faut noter que si n cellules se développent simultanément, réaliser 5000 évaluations de fonction revient à effectuer 5000/n évaluations de fonction par cellule.

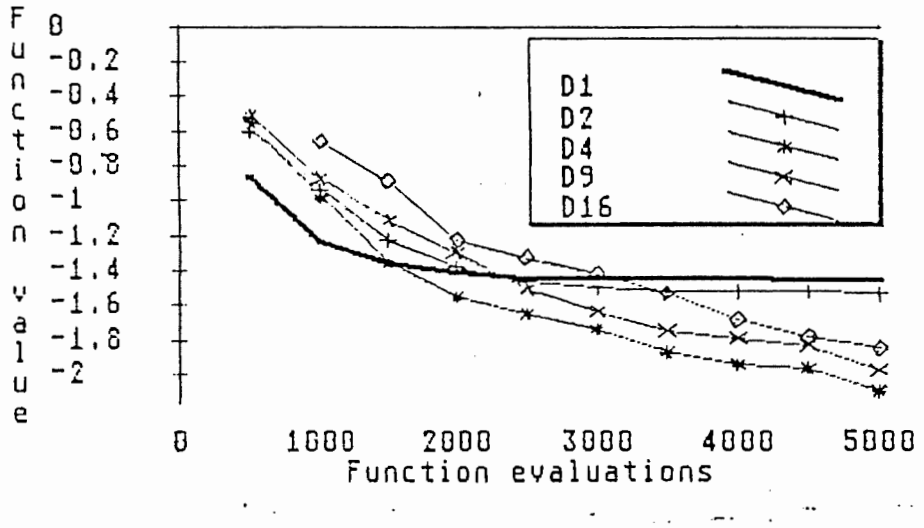


Figure 21 : courbes de performances pour des cellules sans flux de gènes

La courbe D_1 diminue très fortement jusqu'aux environs de 1000 évaluations de fonction pour ensuite se stabiliser, ce qui signifie que la recherche converge avant terme.

En utilisant deux cellules, la convergence prématurée est retardée mais la courbe se stabilise encore. La courbe obtenue avec quatre cellules continue de s'améliorer au-delà de 5000 évaluations de fonction. Les courbes pour 9 et 16 cellules continuent également de s'améliorer au-delà de 5000 évaluations mais elles n'atteignent pas un aussi bon niveau de performances.

Ces expériences confirment qu'il existe un nombre optimal de cellules pour chaque nombre T d'évaluations de fonction. Dans notre exemple, quatre cellules donnent le meilleur résultat après 5000 évaluations. Notons que le nombre optimal de cellules augmente lorsque le nombre total d'évaluations de fonction autorisé augmente. L'expérience a montré que le flux de gènes peut être nuisible à la recherche dans une cellule si le taux de migration est trop élevé et que ses effets sont négligeables si le taux de migration est trop faible. La figure 22 illustre ces propos en donnant les courbes de performances d'un AG à deux cellules avec un taux de migration égal à 5 (D2M5) et un taux de migration égal à 0.2 (D2M0.2). Les meilleures performances sont obtenues avec un taux de migration égal à 1 (D2M1) car avec ce taux de migration, les deux cellules coopèrent pour produire des résultats meilleurs que ceux réalisés avec deux cellules qui se développent indépendamment.

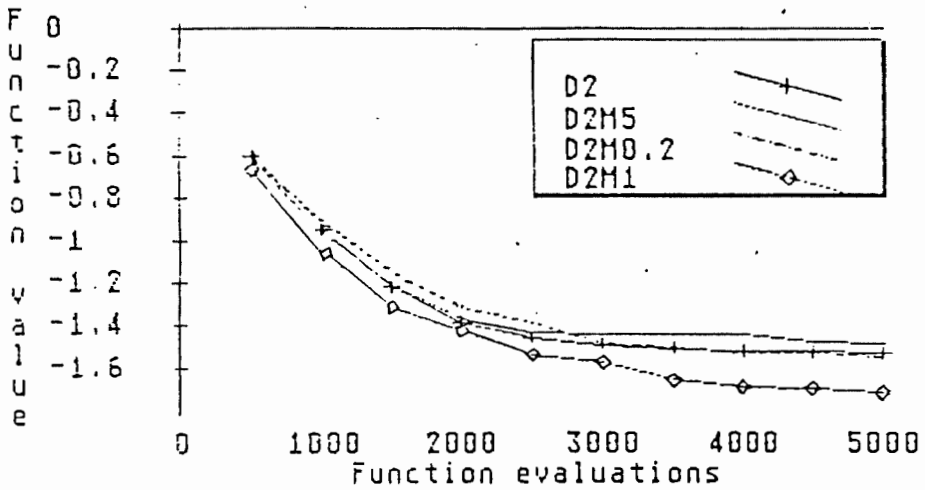


Figure 22 : Courbes de performances pour deux cellules pour différents taux de migration

La figure 23 montre que les performances finales sont meilleures que celles d'un AG traditionnel.

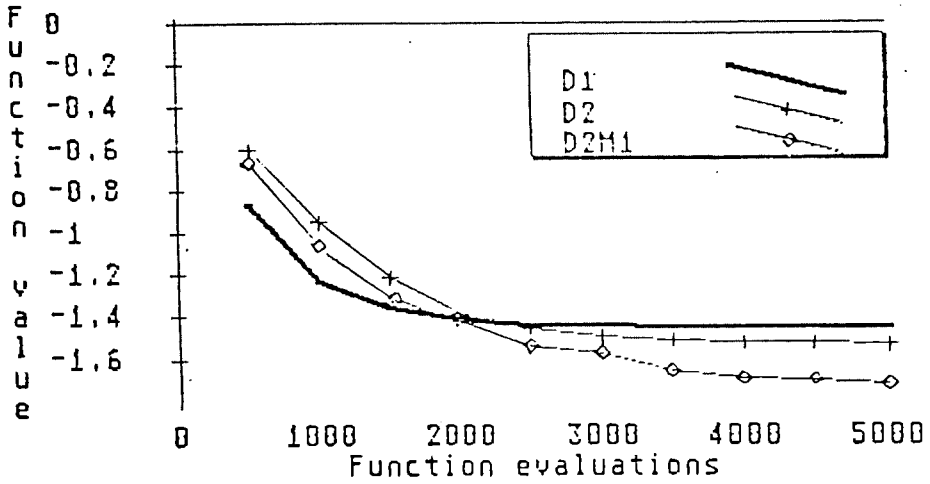


Figure 23 : Courbes de performances pour un AG traditionnel et un AG à 2 cellules

Avec le même taux de migration (m=1), quatre cellules ont le même comportement coopératif et produisent des résultats nettement supérieurs à ceux d'un AG traditionnel.

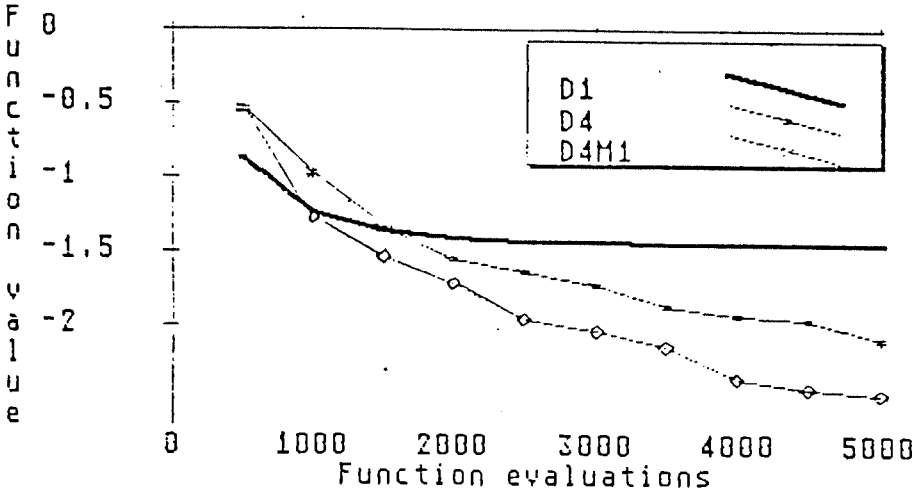


Figure 24 : Courbes de performances pour un AG traditionnel et un AG à 4 cellules

CHAPITRE IV : LE RECUIT SIMULE

IV.1 INTRODUCTION

Dérivé de méthodes de physique statistique, le recuit simulé résout efficacement des problèmes d'optimisation tels que le problème bien connu du voyageur de commerce. En un peu plus de deux années, cette méthode a remporté des succès certains et a connu une industrialisation rapide. Toutefois, elle n'apporte de bons résultats que dans le cadre des problèmes qui peuvent être résolus par des techniques d'amélioration itérative.

IV.2 ANALOGIE ENTRE LA PHYSIQUE STATISTIQUE ET LE RECUIT SIMULE

En physique statistique, on étudie les systèmes formés d'un très grand nombre de particules (molécules, atomes) en interaction. Comme le nombre de ces particules se compte par milliards, la physique statistique ne considère que les propriétés de petits échantillons de quelques centaines de particules. Elle étudie l'évolution de ces échantillons par des simulations. De plus, comme on ne peut pas étudier en continu le mouvement de chaque particule, on détermine un éventail représentatif de toutes les configurations microscopiques du système à partir duquel on calcule les propriétés macroscopiques.

A la fin du XIX^{ème} siècle, le physicien Ludwig BOLTZMANN démontre qu'à l'équilibre thermique à la température T, le temps que passe le système dans une configuration microscopique d'énergie E est proportionnel à exp(-E/T). Lorsque l'on abaisse de manière contrôlée la température d'un ensemble de particules en interaction, le système trouve naturellement, à basse température, un état d'énergie minimale. Il réalise ainsi un processus d'optimisation.

En optimisation combinatoire, on recherche également un échantillon de toutes les configurations en appliquant une transformation qui fait passer aléatoirement de configuration en configuration.

Tout comme la physique statistique, le recuit simulé n'explore donc que certaines configurations représentatives du problème, par exemple les circuits possibles dans le cas du problème du voyageur de commerce. Il utilise l'algorithme de Metropolis (1953) qui permet d'obtenir un échantillon réduit, mais représentatif des configurations et qui permet de simuler sur ordinateur l'évolution aléatoire d'un petit système thermodynamique : on construit des configurations microscopiques du système à l'équilibre thermique, à une température donnée. Or, quand on baisse la température à laquelle on simule le système (d'où le terme de recuit), on n'explore avec l'algorithme de Metropolis que des configurations microscopiques voisines de celle d'énergie minimale, conformément aux lois de la thermodynamique.

Le recuit simulé consiste à partir d'une configuration initiale arbitrairement choisie et à simuler le comportement du système physique à une température initiale donnée. Quand l'équilibre thermique est atteint, on diminue la température et l'on attend à nouveau que l'équilibre thermique s'établisse à la nouvelle température. On répète l'opération et l'on parvient, en principe, à s'approcher de la configuration microscopique d'énergie minimale.

A l'université de Carnegie-Mellon, S. Geman et D. Geman [10] ont trouvé une loi de décroissance de la température qui, partant d'une température infinie, permet d'atteindre la solution exacte après.un nombre exponentiel d'itérations. Cette loi est cependant inexploitable en pratique car on ne peut partir d'une température infinie. D'autre part, la méthode n'est pas plus rapide que les méthodes d'énumération de toutes les solutions. C'est pourquoi en pratique, on utilise des méthodes approchées telles que le recuit simulé pour résoudre les problèmes d'optimisation. Le recuit simulé procure néanmoins une solution quasi-optimale proche de la solution exacte.

IV.3 LA COMPLEXITE DES PROBLEMES

Nous considérons deux problèmes combinatoires pour illustrer le fonctionnement du recuit simulé : le problème du voyageur de commerce (voir précédemment) et le problème de couplage de poids minimal, qui consiste à relier deux à deux tous les points d'un ensemble donné, de telle sorte que la somme des longueurs des segments soit minimale. Ces problèmes d'optimisation sont susceptibles de se traduire par une fonction "coût" analogue à l'énergie d'un système physique, prenant une valeur particulière pour chaque configuration : la longueur d'un circuit dans le premier problème, et la somme des longueurs dans le second. On recherche les configurations pour lesquelles la valeur de cette fonction est optimale (minimale).

En général, le nombre de configurations des problèmes traités augmente rapidement en fonction de la "taille" des problèmes : le nombre de villes dans le cas du voyageur de commerce, et le nombre de points à apparier dans le second exemple. Dans le premier cas, si N est le nombre de villes, le nombre de circuits possibles est égal à [(N-1)!]/2 soit (N-1)(N-3)(N-5)...5*3*1/2; dans le second, le nombre d'appariements est égal à (N-1)(N-3)...5*3*1. Dans les deux cas, la méthode consistant à rechercher la solution en énumérant toutes les configurations est rédhibitoire quand la taille des problèmes est supérieure à 10 ou plus; on doit donc utiliser des méthodes de recherche, ou algorithmes, plus élaborées que l'on caractérise par leur complexité c'est-à-dire le nombre d'opérations à effectuer pour trouver la solution d'un problème de taille donnée dans la situation la plus défavorable.

On peut classer les problèmes en fonction de la complexité de leur algorithme. Le problème de couplage de poids minimal est un exemple de problème "facile" : on connaît un algorithme qui donne une solution exacte en effectuant un nombre d'opérations (nombre de pas) proportionnel à N^3. On dit que le problème est polynomial. Le problème du voyageur de commerce est l'exemple d'un problème "difficile" : dans toutes les méthodes exactes que l'on a conçues pour déterminer la tournée optimale du voyageur de commerce, le nombre de pas de calcul augmente exponentiellement en fonction du nombre N de villes. Ces méthodes

reviennent toutes à effectuer une énumération partielle de l'ensemble des configurations, de sorte que l'on ne sait traiter que des problèmes comportant quelques centaines de points (le plus gros problème du voyageur de commerce résolu exactement jusqu'à présent comporte 532 villes).

De nombreux problèmes d'optimisation comportent parfois des milliers de variables. C'est pourquoi on met en oeuvre des méthodes approchées rapides appelées heuristiques, mais les approximations simplificatrices nuisent à la qualité de la solution calculée par rapport à la solution exacte. Le choix d'une méthode simplifiée sera donc dicté par un compromis entre la vitesse d'exécution du programme sur ordinateur et la qualité du résultat.

IV.4 L'AMELIORATION ITERATIVE

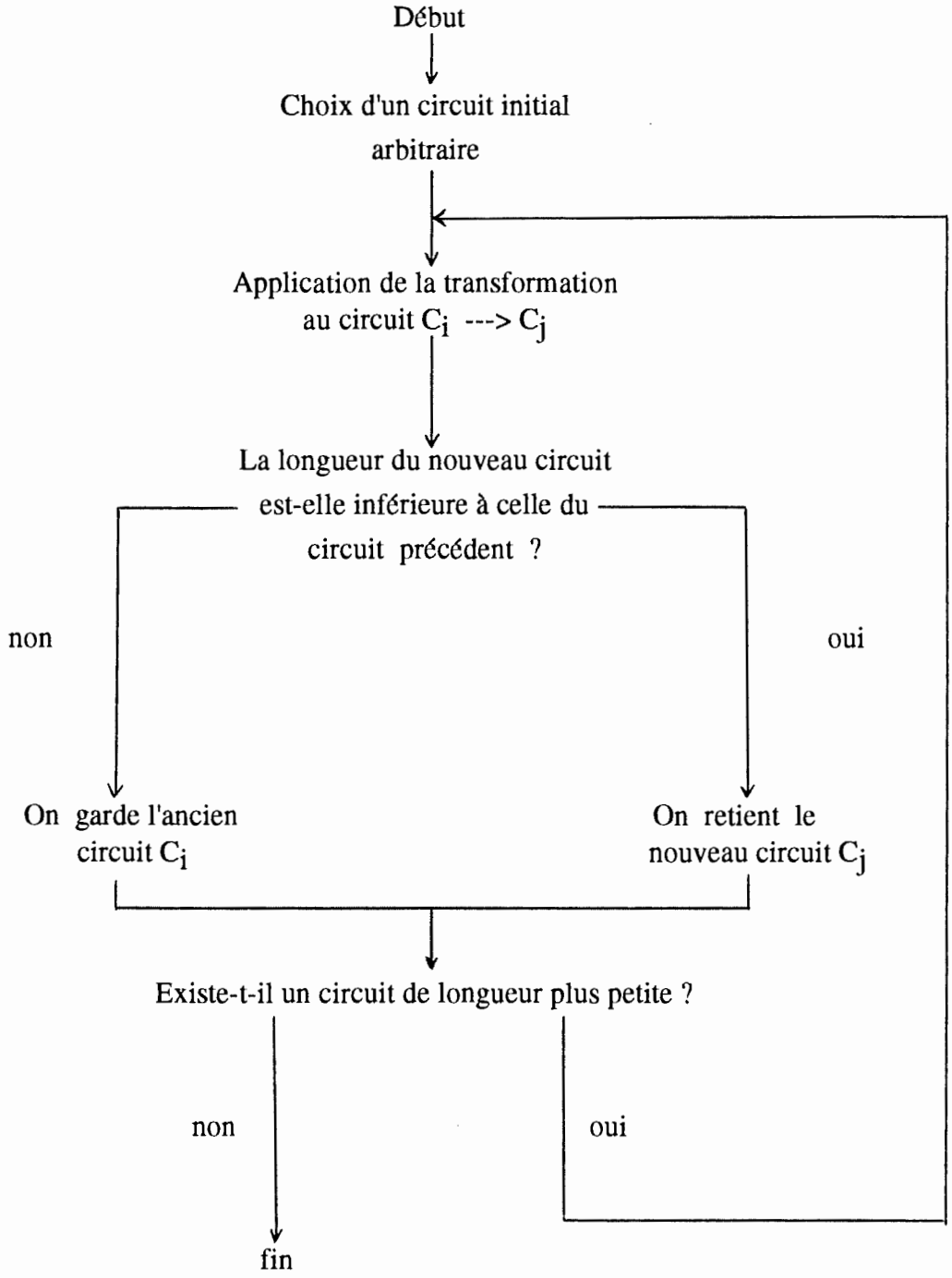
L'amélioration itérative est une heuristique qui porte le germe du recuit simulé. Il s'agit d'une famille de méthodes simples à mettre en oeuvre et qui produisent des solutions approchées en explorant partiellement l'ensemble des configurations possibles - les différents circuits, dans le problème du voyageur de commerce. Elles présentent l'inconvénient d'aboutir parfois à des solutions de coût fort supérieur à celui de la solution exacte du problème.

Les méthodes d'amélioration itérative consistent à partir d'une configuration, et à modifier celle-ci par une transformation élémentaire; on calcule ensuite la variation de la fonction de coût résultant de cette modification; si celle-ci est positive, c'est que la nouvelle solution est moins bonne que la précédente : elle est refusée; dans le cas contraire, la modification a produit une amélioration : elle est donc acceptée. Le processus est itéré jusqu'à ce qu'aucune transformation ne puisse réduire la valeur de la fonction de coût. Ce processus conduit très souvent à des solutions qui sont sous-optimales car il ne permet de trouver que des minima locaux et n'offre donc aucune garantie d'aboutir à un minimum global.

Si on l'applique au problème du voyageur de commerce, l'amélioration itérative consiste à partir d'un circuit choisi arbitrairement puis à lui appliquer une transformation afin d'obtenir un nouveau circuit. Si la longueur du nouveau circuit obtenu est inférieure à la longueur du circuit initial, on "oublie" le circuit initial et l'on retient le nouveau circuit, à partir duquel on répète l'opération. Par contre, si la longueur du nouveau circuit est supérieure à la longueur du circuit initial, on ne retient pas le nouveau circuit et l'on applique à nouveau la transformation afin d'obtenir un autre circuit dont on compare à nouveau la longueur à celle du circuit initial, et ce jusqu'à ce que l'on trouve un circuit de longueur inférieure.

Une transformation possible pour ce problème consisterait à supprimer deux arcs et à les remplacer par deux autres arcs. On répète cette triple opération - application d'une transformation, comparaison des longueurs, transition vers le nouveau circuit ou rejet du circuit trouvé - jusqu'à ce que plus aucune transformation ne produise de circuit plus court.

Généralement, cette procédure ne conduit pas au circuit le plus court car on finit "piégé" dans un minimum local, c'est-à-dire dans un circuit dont tous les circuits voisins, accessibles par la transformation utilisée sont de longueur supérieure.



IV.5 L'ALGORITHME DE METROPOLIS [9] [10] [11]

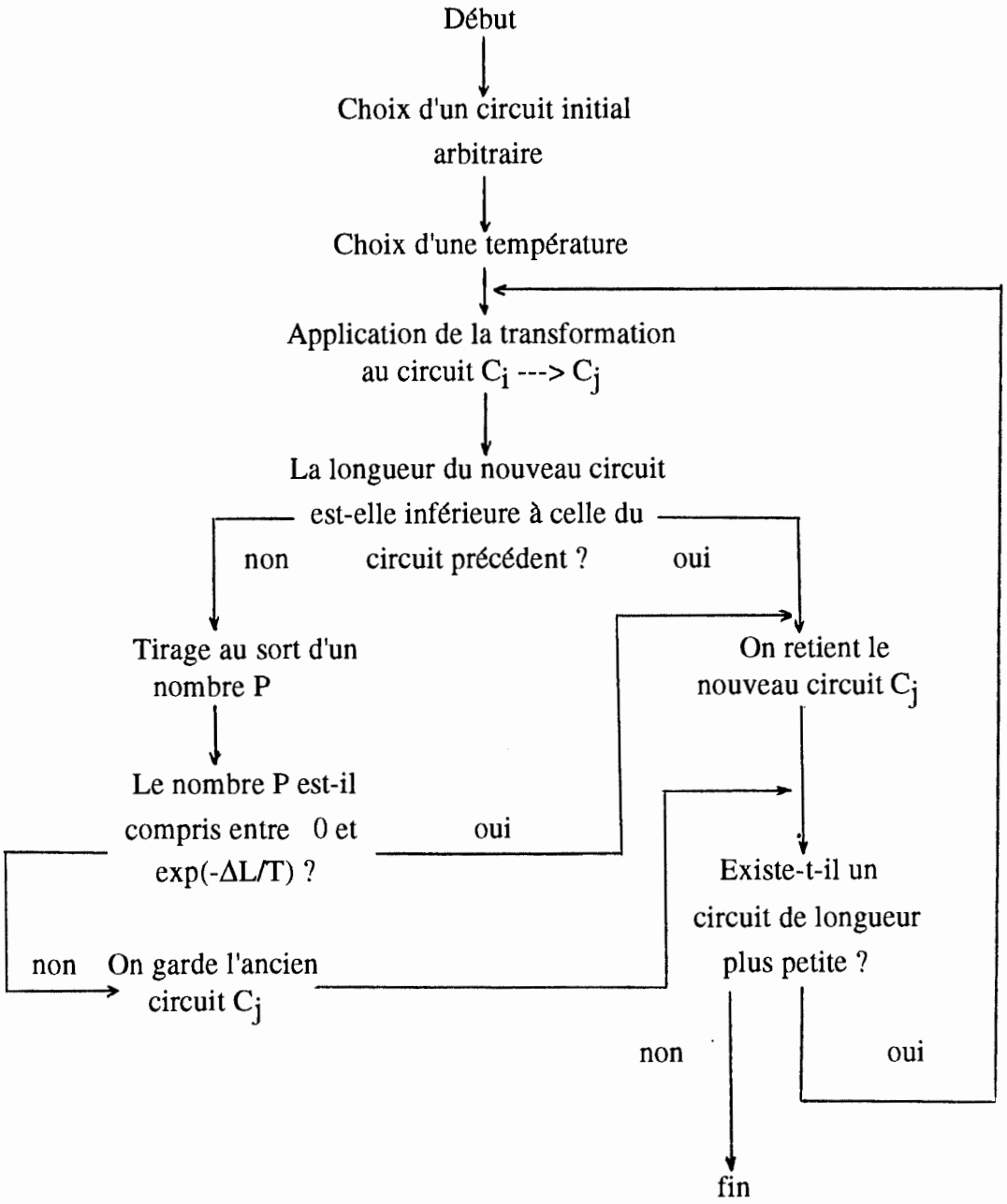
L'algorithme de Metropolis est une méthode qui contrairement à l'algorithme d'amélioration itérative permet de sortir des minima locaux en utilisant un paramètre T appelé température. Il sert précisément à tirer au

hasard une suite de configurations microscopiques en respectant les proportions de Boltzmann relatives à l'équilibre à une température donnée. Dans le cas de l'amélioration itérative, nous avons vu que l'évolution s'effectue toujours vers les configurations de coût décroissant, tandis que l'algorithme de Metropolis permet parfois des transitions vers des configurations de coût supérieur. De manière plus précise, l'algorithme d'optimisation est le suivant : partant d'une configuration microscopique C_i, d'énergie E_i, on fait subir au système une modification élémentaire. Si la transition a pour effet de diminuer l'énergie du système lors du déplacement, on l'accepte. Par contre, si elle a pour effet d'augmenter l'énergie (fonction de coût), on ne la refuse pas systématiquement : la probabilité de l'accepter est de $\exp(-\Delta E/T) = \exp(-(E_j-E_i)/T)$ où T est la température et ΔE la variation de la fonction de coût résultant de la transformation élémentaire effectuée. Pour un système physique, la température conditionne le nombre d'états accessibles, et, si elle est abaissée de façon suffisamment lente pour que l'on ne s'éloigne pas de l'équilibre thermodynamique, elle permet d'atteindre un état d'énergie minimale. Pour un problème d'optimisation, la température n'a pas de signification immédiate : en fait, elle constitue un paramètre de contrôle qui permet de balayer l'espace des solutions possibles de façon à atteindre l'optimum avec une grande probabilité.

A basse température (T tend vers 0), $\exp(-\Delta E/T)$ tend vers 0 et donc la probabilité d'accepter des transformations qui augmentent l'énergie devient nulle. L'algorithme de Metropolis est alors identique à l'algorithme d'amélioration itérative, et l'on risque de finir piégé dans des minima locaux, comme on l'a vu précédemment.

Quand la température est élevée (lorsque T tend vers l'infini), $\exp(-\Delta E/T)$ tend vers 1 donc la plupart des mouvements sont acceptés. Dès lors, quand la température n'est pas nulle, la marche aléatoire que l'algorithme de Metropolis produit permet d'explorer le graphe des configurations en favorisant les configurations d'énergie inférieure. En outre, plus la température est faible, plus les configurations de faible énergie sont favorisées. Dans le cas du voyageur de commerce, par exemple, les configurations sont les circuits et le coût est la longueur du circuit.

Pratiquement, à partir d'une configuration C_i, on applique une transformation produisant une configuration C_j. On retient toujours cette nouvelle configuration C_j si la longueur L' du circuit est inférieure à la longueur L de la configuration initiale C_i. Par contre, si la longueur L' est supérieure à la longueur L, on choisit au hasard un nombre P compris entre 0 et 1; si ce nombre est compris entre 0 et exp(-(L'-L)/T) on retiendra le circuit. De la sorte, on construit pas à pas une marche aléatoire sur le graphe.



IV.6 LE RECUIT SIMULE

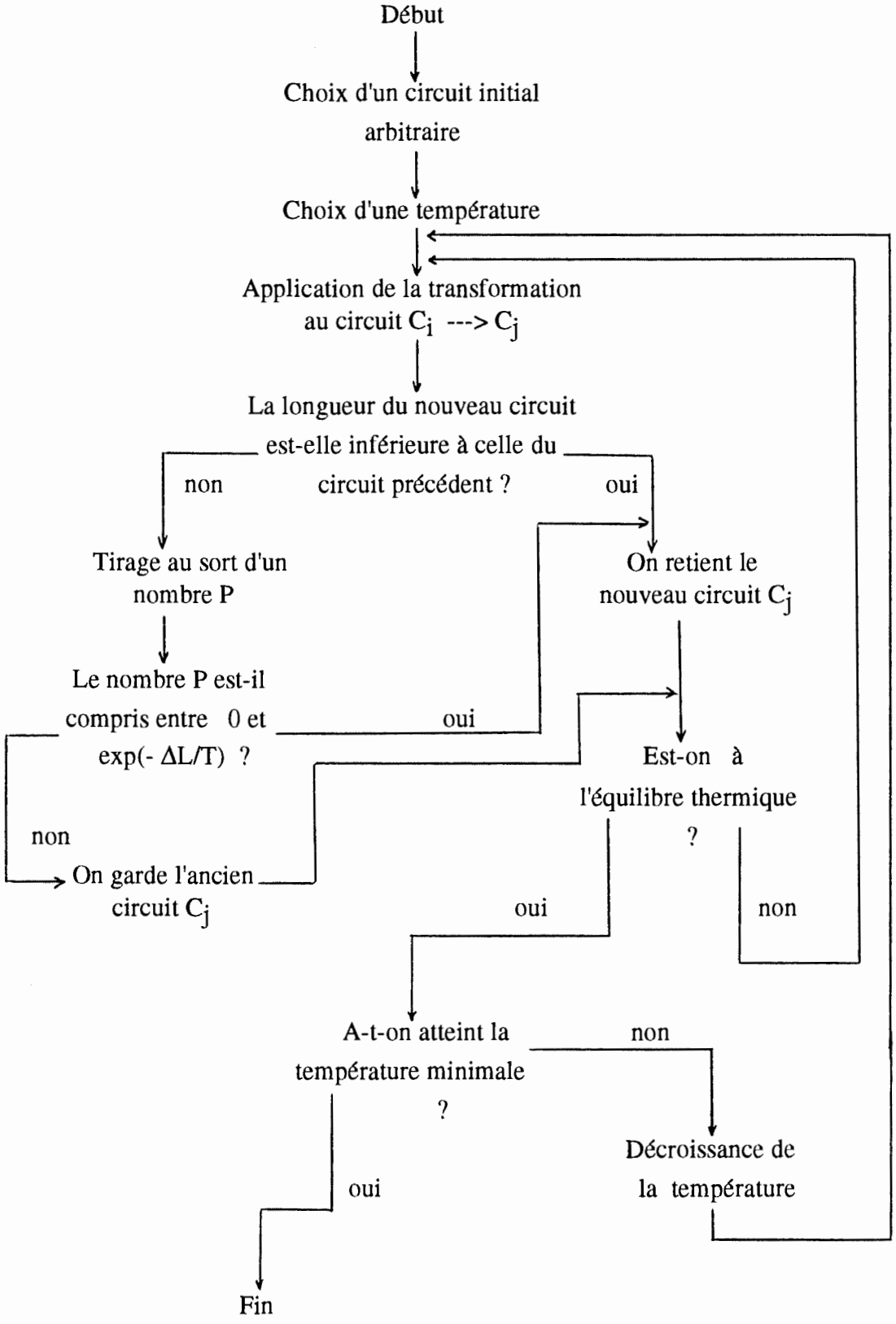
IV.6.1 PRINCIPE DE LA METHODE

En terme de métallurgie, l'amélioration itérative est l'équivalent d'une trempe (abaissement brutal de la température), qui produit un solide amorphe, alors que l'abaissement lent de la température constitue un recuit, qui permet d'obtenir un solide cristallin d'énergie minimale.
La méthode du recuit simulé consiste à effectuer une marche aléatoire dans l'ensemble des configurations, grâce à l'algorithme de Metropolis tout en réduisant progressivement la température. De la sorte, on explore des configurations de coût globalement décroissant.
Pour résoudre un problème par la méthode du recuit simulé, nous avons besoin des éléments suivants :

- une configuration et une température initiales
- la règle de transformation des configurations (liste des mouvements élémentaires permis)
- une fonction de coût (énergie du système)
- le schéma de décroissance ou profil de la température
- le nombre d'itérations de la transformation à chaque température
- le critère d'arrêt

En pratique, on partira d'une configuration arbitraire, mais pas trop absurde, et l'on effectuera, par application de l'algorithme de Metropolis, une "relaxation" à une température d'abord élevée, c'est-à-dire que l'on effectuera une série de transitions entre circuits respectant les proportions de Boltzmann. Quand l'équilibre thermique est atteint, après un nombre de transitions qui dépend de la taille du problème, on réduit légèrement la température et l'on fait une nouvelle relaxation à la nouvelle température. On répète le cycle de refroidissements et relaxations. Finalement, avec des temps de calcul suffisamment longs et une température finale faible, on obtient des solutions proches de la solution exacte sans avoir été piégé dans un minimum local. Comme le but est d'obtenir une solution proche de la solution exacte en un temps aussi court que possible, le nombre de transitions caractérisant chaque relaxation doit être limité : l'algorithme deviendra nécessairement

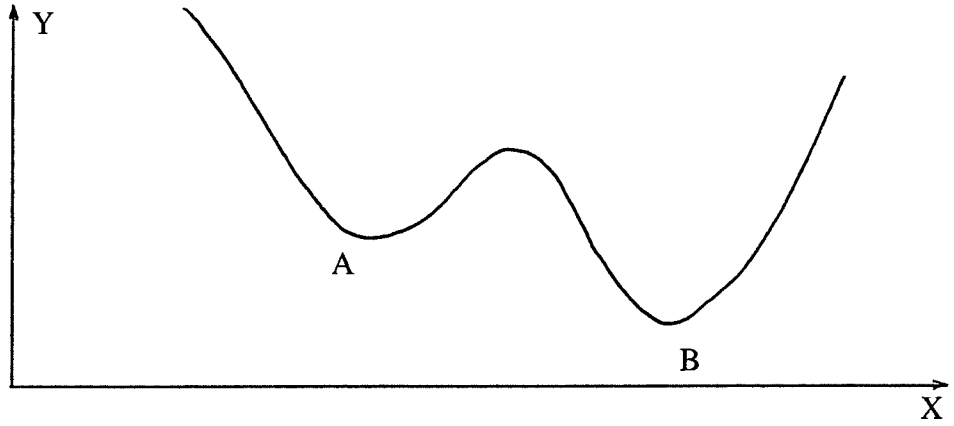
approximatif et ne procurera qu'une solution quasi-optimale qui est toutefois proche de la solution exacte.



IV.6.2 CHOIX DES TEMPERATURES

La température initiale doit être élevée car si la solution initiale est prise complètement au hasard, il y a très peu de chances pour qu'elle soit voisine de l'optimum. On partira donc d'une température élevée pour "oublier" la situation initiale et visiter l'espace des états; en revanche la méthode du recuit simulé peut être utilisée pour affiner une solution considérée à priori comme satisfaisante : dans ce cas, on partira d'une température modérée.

On peut comprendre la raison fondamentale du recuit - se placer à haute température, puis refroidir progressivement le système en imaginant la situation suivante :

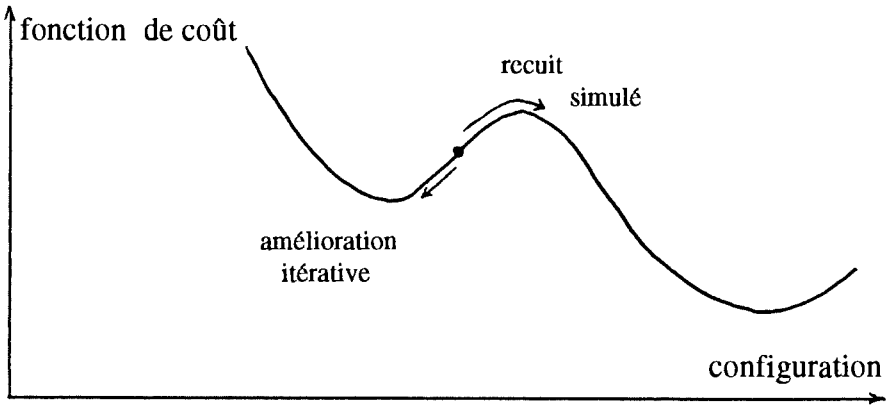


L'axe X représente tous les points de l'espace et l'axe Y la valeur de chaque point. Le but est de trouver le point de l'espace avec la plus petite valeur, en l'occurence B. Une bille, située en un point choisi arbitrairement, cherche à descendre vers le creux le plus profond. Si nous donnons une secousse au système, la bille aura plus de chances de se déplacer de A vers B que vice-versa étant donné qu'il y a un obstacle plus bas à franchir du côté de A. Donc, si la secousse est légère, une transition de A vers B sera nettement plus probable qu'une transition de B vers A. En d'autres mots, cela signifie que la bille ne pourra jamais remonter le col qui la ferait passer dans le creux le plus profond. La bille accepte de remonter si elle n'a pas à faire un effort trop pénible.

D'autre part, si le choc est violent, la bille passera fréquemment l'obstacle, mais il lui sera aussi facile de se déplacer de B vers A que de A vers B. Un bon compromis serait de commencer par appliquer une secousse

violente au système et de la diminuer graduellement. C'est exactement ce qui se passe pour la technique de recherche du recuit simulé : les températures élevées correspondent aux secousses violentes, à une grande vigueur de la bille qui accepte de remonter beaucoup à chaque pas. D'où l'on voit la nécessité d'une température initiale élevée pour éviter d'être piégé dans un minimum local, la bille ayant assez de "force" pour remonter la pente et atteindre le creux le plus profond. Cette température initiale sera ensuite abaissée artificiellement (simulée) jusqu'à une température où plus aucune transition n'est possible.

Donc, l'exploration initialement énergique de l'espace (chaque point a quasiment la même chance d'être visité) sera graduellement remplacée par l'exploitation (le nouveau point à visiter résidera dans le voisinage direct du point considéré).



L'amélioration itérative classique conduit à des minima locaux tandis que le recuit simulé permet de "remonter la pente".

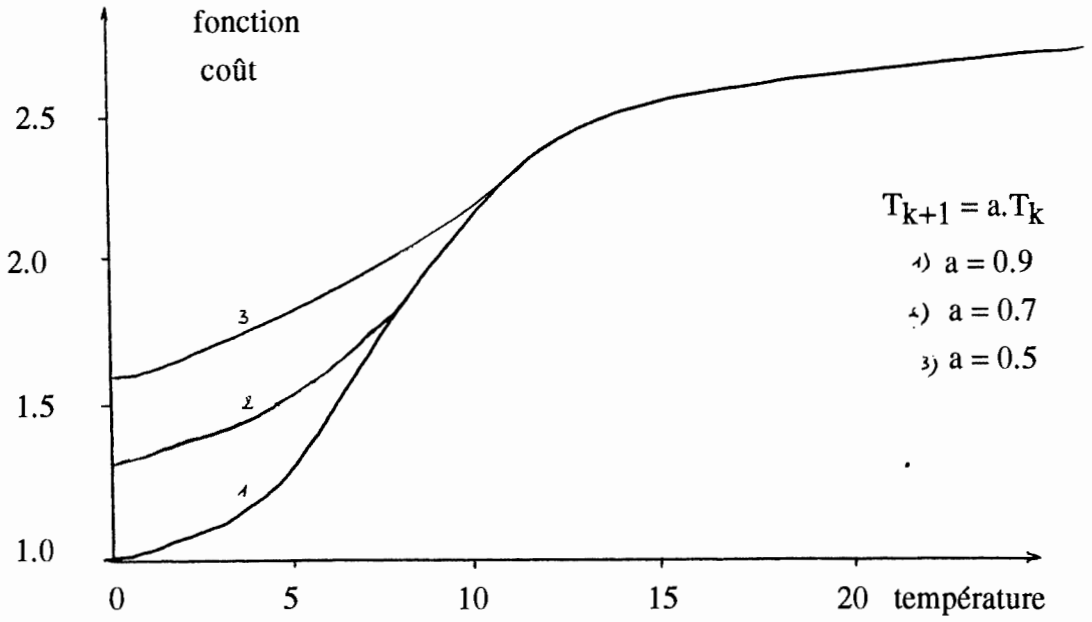
Pour le problème du voyageur de commerce, le recuit simulé élimine rapidement la grande majorité des circuits possibles et, dès que la température à laquelle on applique l'algorithme de Metropolis est faible, n'explore que les circuits les plus courts.

Lors de l'exécution du recuit simulé, les soucis de rapidité de calcul contraignent néanmoins à raccourcir la marche aléatoire par le choix d'une valeur réduite de la température initiale. Un bon critère expérimental consiste à choisir une température initiale telle que la proportion de transitions qui sont retenues du fait de la transformation

choisie soit égale à 50 pour cent. Le deuxième paramètre important pour la mise en oeuvre du recuit simulé est le profil de la température c'est-à-dire les paliers de température, lors desquels on explore l'ensemble des circuits par répétition de la transformation.

Un bon compromis expérimental entre la rapidité de calcul et la qualité du résultat est obtenu par décroissance géométrique de la température du type $T_{k+1} = a \, .T_k$ avec a<1, ce qui signifie que la température d'un palier est égale à la température du palier précédent multiplié par un facteur donné, 0.93 par exemple.

La figure ci-dessous montre l'évolution de la valeur moyenne de la fonction de coût durant le recuit, pour trois lois de variation de la température en fonction du temps : un refroidissement trop rapide conduit à un optimum local, alors qu'une valeur de a voisine de 0.9 permet d'atteindre un optimum global.



Selon le problème traité, on effectuera entre 30 et 50 paliers et, à chaque palier, le nombre de transitions dépendra de la taille du problème et du temps de calcul que l'on veut lui consacrer. Le schéma de recuit - c'est-à-dire la façon dont on fait diminuer la température, le nombre d'itérations par palier et le critère d'arrêt - est certes important, mais c'est surtout la transformation choisie qui fixe l'efficacité de la résolution.

Dans le cas du voyageur de commerce, cette efficacité dépend donc de la nature de la transformation appliquée au circuit lors des transitions. En effet, quand le coût du nouveau circuit obtenu est supérieur au coût du circuit dont on part, la probabilité d'accepter le nouveau circuit et d'effectuer la transition sera égale à la valeur de $[\exp(-(E_j-E_i)/T)]$; de ce fait, quand le voisinage d'un circuit contient une proportion importante de circuits de coût bien supérieur, le nombre de transitions non-retenues est d'autant plus important que la température est faible. Une transition refusée étant synonyme de temps perdu, il faut utiliser une transformation pour laquelle le voisinage de tout circuit ne contienne que des circuits de coût peu différent.

En toute généralité, la mise en oeuvre de cette stratégie est délicate, mais pour les problèmes d'optimisation combinatoires de nature géographique, comme le problème du voyageur de commerce, avec des fonctions coûts qui augmentent avec la distance, cette tâche n'est pas difficile. Il suffit de décomposer le domaine en sous-domaines, puis d'appliquer la même transformation que celle décrite précédemment, en imposant en outre que les extrémités des deux arcs considérés soient dans des sous-domaines adjacents. Avec cette amélioration de la méthode, on obtient de bonnes solutions pour des problèmes de grande taille, avec des temps de calcul simplement proportionnels à la taille N du problème.

IV.7 PRESENTATION DE L'ALGORITHME

Soit f, la fonction objectif à minimiser, a : un coefficient de réduction de la température ($0<a<1$), change : une variable booléenne mise à vrai aussi longtemps que des modifications sont apportées à la fonction objectif du problème.
On exécute des cycles de rep générations aléatoires de solutions voisines à une température constante. La procédure s'arrête si durant un cycle complet, il n'y a eu aucun changement apporté à la fonction objectif (c'est-à-dire quand la variable change a pour valeur 'faux'). Le choix du nombre rep de répétitions a une grande influence sur la qualité de la solution.

Algorithme :

Initialiser t à une valeur élevée de façon appropriée

Choisir une solution possible S et évaluer la fonction objectif f(S)

change := vrai

```
while change do
begin
    change := faux
    repeat rep times
    begin
        transformer S en S'
        Δ = f(S')-f(S)
        if Δ < 0 then go to accept
        p(Δ) = exp (- Δ /t)
        Générer une variable aléatoire x (x [0,1])
        if x < p( Δ) then go to accept
                    else go to exit
        accept S := S'
                f(S') := f(S) + Δ
        if Δ<> 0 then change := vrai
        exit
    end
    t := t * a
end
end.
```

CHAPITRE V : LE PROBLEME DU VOYAGEUR DE COMMERCE - RESULTATS EXPERIMENTAUX

Nous décrivons dans les pages qui suivent les solutions obtenues par la méthode du recuit simulé (pour des PVC à 12, 25, 50 et 100 villes) et par algorithmes génétiques (PVC à 12, 25, 50 et 70 villes).
Tous les résultats cités représentent la moyenne d'une dizaine d'essais; ces derniers ont été réalisés sur un PC - 386 avec une mémoire de 4 MB.

V.1 RECUIT SIMULE

En se basant sur le tableau des résultats de la figure 5.1, nous pouvons tirer les conclusions suivantes :

PVC à 12 villes

Pour des PVC avec un nombre de villes relativement faible (plus ou moins 10), des coefficents de température situés entre 0.4 et 0.8 donnent les meilleures performances. En effet, ces valeurs conduisent à une solution optimale en un minimum de paliers de température (de 10 à 30 itérations). Par contre, si on utilise des coefficients de température plus importants (de l'ordre de 0.8 - 0.93), il faut effectuer une centaine d'itérations avant d'arriver à un résultat similaire.
Par conséquent, le temps de traitement est beaucoup plus long. Un petit nombre de répétitions (REP) par palier de température s'avère ici suffisant (500 - 1.000 répétitions).

PVC à 25 villes

Avec l'augmentation du nombre de villes, nous constatons que des coefficients de l'ordre de 0.4 - 0.8 entraînent la convergence vers un minimum local. Même en faisant varier le nombre de répétitions par palier de température, on ne parvient pas à sortir de ces minima locaux; on n'arrive donc jamais à une solution optimale. Des paramètres plus proches de l'unité (0.93) permettent de franchir ces minima locaux et de trouver le circuit le plus court (longueur = 40.59). Après de nombreux essais, le nombre de répétitions par palier s'avère concluant aux alentous de 5.000. Au-delà de cette valeur (par exemple 10.000),

aucune amélioration n'a été constatée. En-deçà (500-1.000), la solultion converge vers un minimum local. Toutefois, plus le coefficient de température se rapproche de 1, plus il faudra de paliers de température, pour parvenir à la meilleure solution. En effet, une moyenne de 80 itérations est nécessaire pour un coefficient de température égal à 0.93.

PVC à 50 villes

Pour des PVC à 50 villes, les résultats précédents se confirment étant donné que les solutions optimales sont obtenues en utilisant pour les paramètres COEFF des valeurs comprises entre 0.8 et 0.93 et pour le paramètre REP une valeur située dans l'intervalle 5.000 - 10.000. La solution obtenue en utilisant COEFF = 0.8 ne nécessite en moyenne qu'une trentaine d'itérations mais ne permet pas d'atteindre la solution optimale puisque l'on reste bloqué dans des minima locaux. Comme précédemment, toute valeur de REP supérieure à 10.000 n'apporte pas d'améliorations sensibles.

PVC à 100 villes et plus

Plus le coefficient de température se rapproche de 1, meilleures seront les solutions obtenues lorsque le nombre de villes augmente. Dans nos exemples, le paramètre COEFF égal à 0.8 donne une longueur de 108.63 après 418 itérations et COEFF = 0.93 donne une longueur de 108.21 après 845 itérations. Les résultats confirment que les paramètres REP compris entre 5.000 et 10.000 apportent une solution optimale.

La figure 5.2 représente le "circuit initial pas trop absurde" pour un PVC à 100 villes et la figure 5.3 décrit le circuit de longueur minimale obtenu après 845 paliers de température à raison de 5.000 répétitions par palier. Si l'on se réfère à la longueur du circuit initial (145.20), le gain obtenu par le traitement du recuit est de 25.5 %.

A la figure 5.4, nous constatons que la longueur devient optimale pour des coefficients de température se rapprochant de l'unité. Tout comme en métallurgie, on constate donc que si on diminue lentement la température, le système se met dans son état de moindre énergie.

La figure 5.5 décrit pour chaque coefficient de température, l'évolution de la longueur en fonction du nombre de répétitions par palier. Le fait d'augmenter le nombre de répétitions au-delà de 10.000 ne modifie pas le résultat obtenu. Par conséquent, le meilleur compromis entre le résultat et le temps de traitement est obtenu avec un coefficient REP = 5.000.

Fig. 5.1 : TABLEAU DES RESULTATS OBTENUS PAR LA METHODE DU RECUIT SIMULE

	COEFF	REP	STOP	LONG.	NOMBRE ITER
N = 12	0.4	500	100	25.77	10
	0.4	1000	100	25.77	9
	0.6	500	100	25.77	16
	0.6	1000	100	25.77	12
	0.6	5000	100	25.77	17
	0.8	500	100	25.77	30
	0.8	1000	100	25.77	31
	0.8	5000	100	25.77	33
	0.93	500	100	25.77	84
	0.93	1000	100	25.77	90
	0.93	5000	100	25.77	17
N = 25	0.4	500	100	42.22	11
	0.4	1000	100	41.82	8
	0.6	500	100	43.12	13
	0.6	1000	100	42.85	12
	0.6	5000	100	42.19	12
	0.8	500	100	42.66	22
	0.8	1000	100	41.18	30
	0.8	5000	100	41.18	27
	0.93	500	100	41.72	67
	0.93	1000	100	41.18	64
	0.93	**5000**	**100**	**40.59**	**74**
	0.93	10000	100	40.59	79
N = 50	0.6	1000	100	65.51	15
	0.6	5000	100	62.85	16
	0.6	25000	100	61.21	20
	0.7	1000	100	64.23	100
	0.7	5000	100	61.21	37
	0.8	1000	100	63.28	27
	0.8	5000	100	60.28	26

	0.8	25000	100	60.18	25
	0.93	1000	100	62.63	76
	0.93	**5000**	**100**	**60**	**78**
	0.93	10000	100	60	80
	0.93	25000	1000	59.8	136
N = 100	0.6	1000	1000	132.85	158
	0.6	5000	1000	109.63	158
	0.8	1000	1000	122.58	382
	0.8	5000	1000	108.63	418
	0.93	1000	1000	121.64	645
	0.93	**5000**	**1000**	**108.21**	**845**

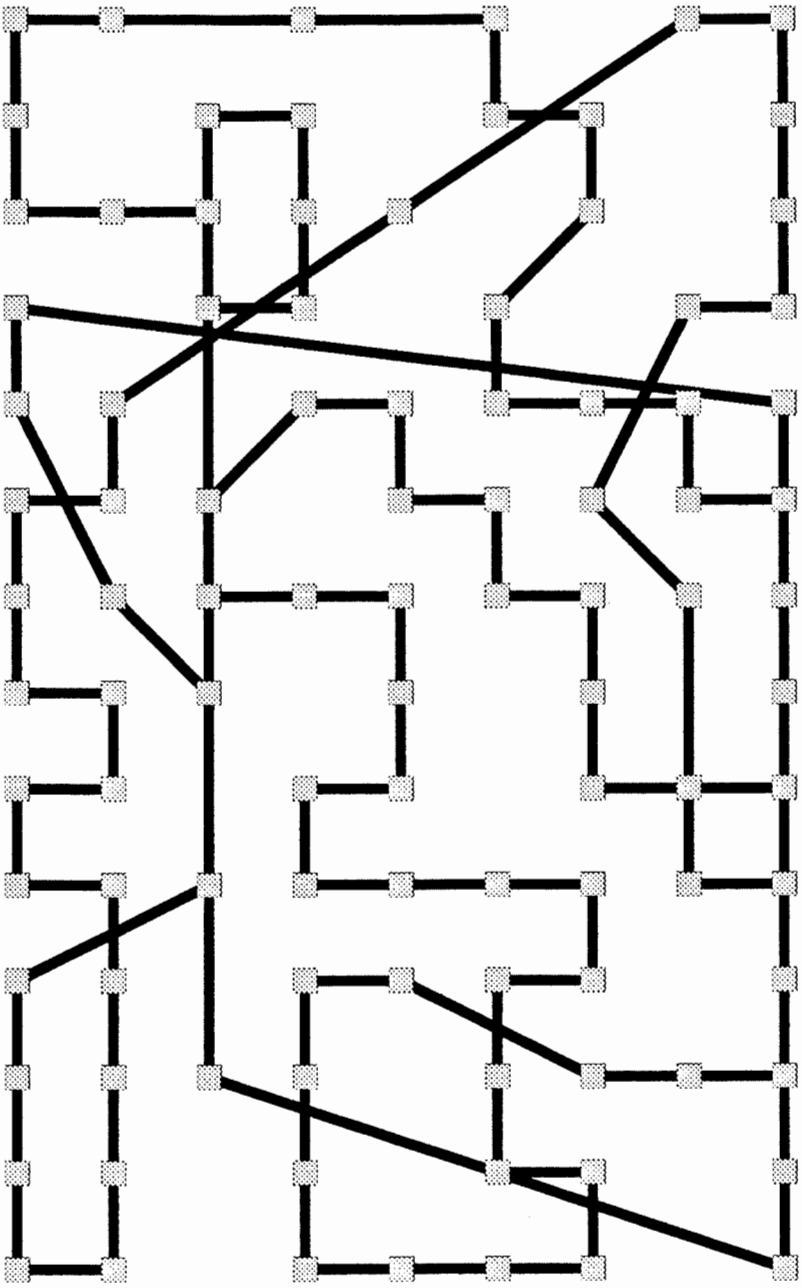


Fig.5.2

Recuit simulé : circuit initial

N = 100 villes
COEFF = 0.93
REP = 5000
LONGUEUR : 145.20

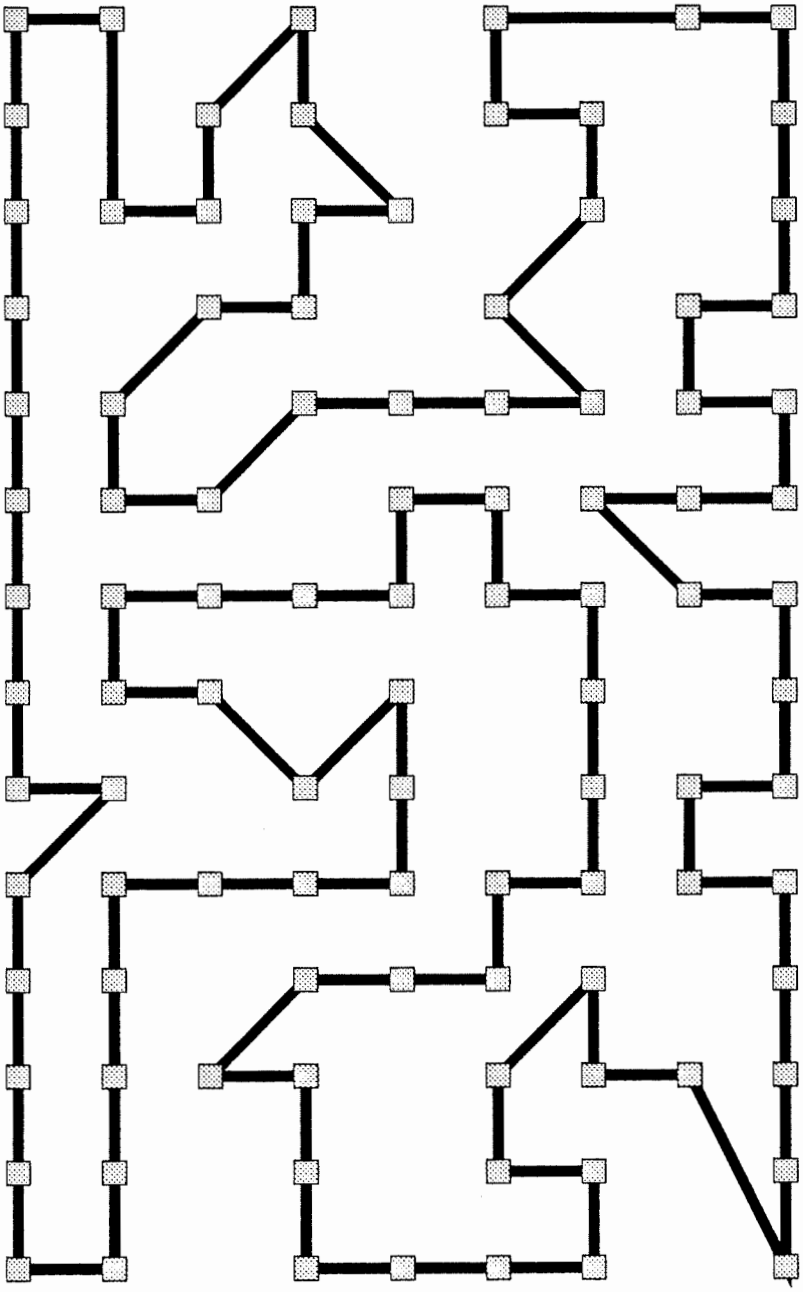


Fig.5.3

Recuit Simulé : circuit final

N = 100 villes
COEFF = 0.93
REP = 5000
LONGUEUR : 108.21

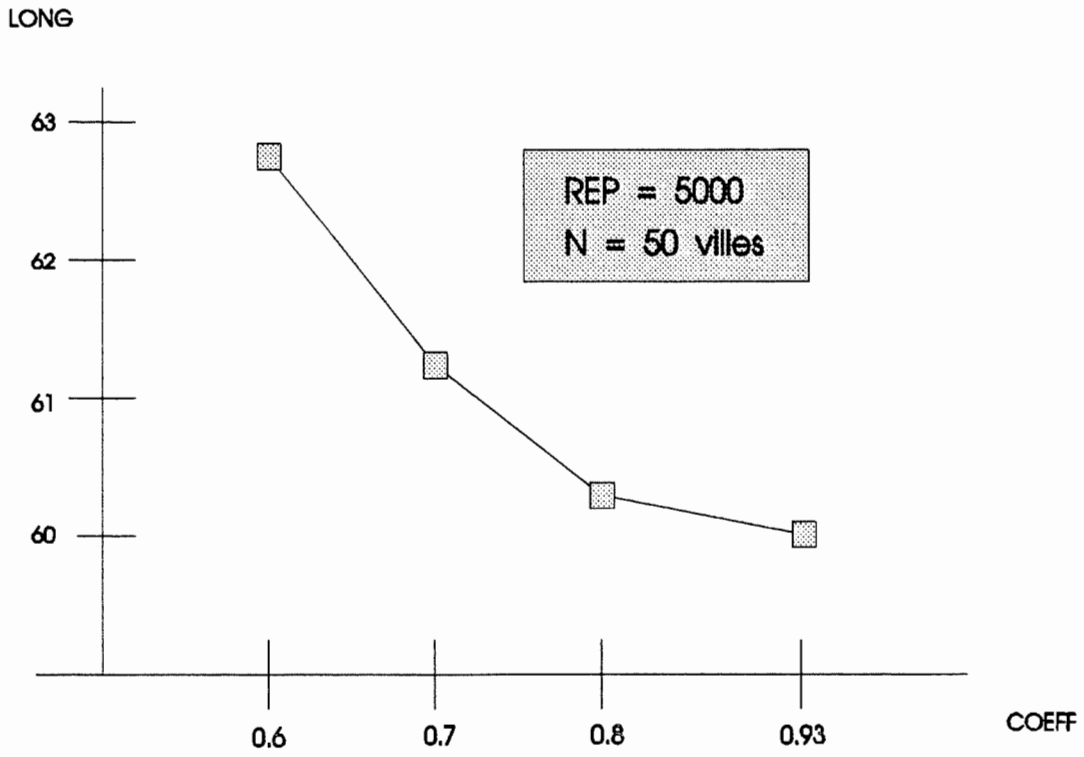


Fig.5.4 : Evolution de la longueur du circuit pour différentes lois de décroissance de la température

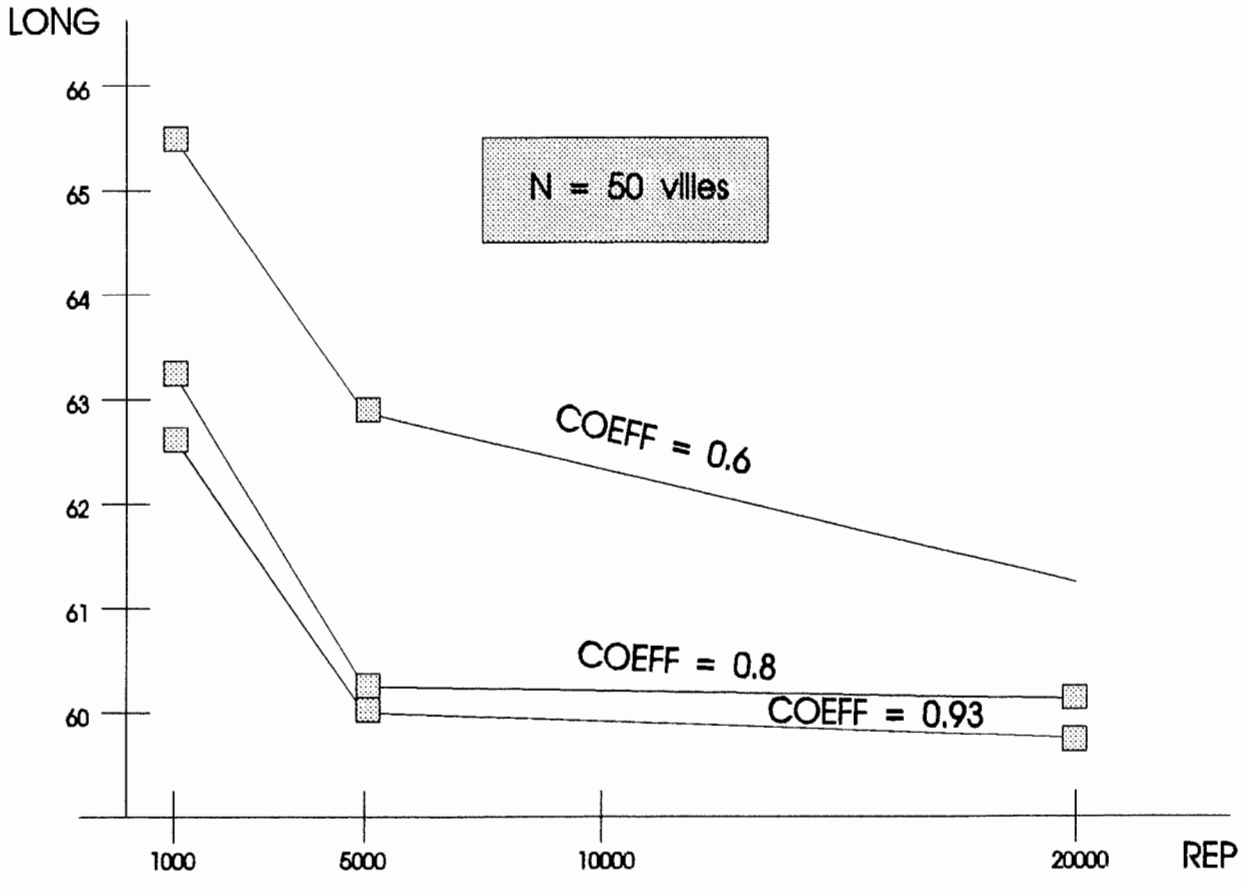


Fig.5.5 : Evolution de la longueur du circuit en fonction du nombre de répétitions par palier de température

LISTING DU PROGRAMME "RECUIT SIMULE"

```
PROGRAM RECUITSIM;

USES CRT;

TYPE        TAB=ARRAY[1..100] OF ARRAY[1..100] OF REAL;
            VECT=ARRAY[1..100] OF INTEGER;
            VECREEL=ARRAY[1..100] OF REAL;

CONST       COEFF=0.93;
            REP=5000;
            STOP=1000;

VAR         I,J,N,Z,GI,DI,NBITERATION,DIM1,DIPGI:INTEGER;
            DMIN,LONG,DL,SOMDL,VAL,ALEAT,NBRE:REAL;
            A:TAB;
            C,COLD:VECT;
            X,Y:VECREEL;
           CHANGE:BOOLEAN;
            T:REAL;


BEGIN

{INITIALISATION}

RANDOMIZE;

FOR I:=1 TO 100 DO
  BEGIN
    FOR J:=1 TO 100 DO
      A[I,J]:=0;
    C[I]:=0;
    X[I]:=0;
    Y[I]:=0;
```

```
  END;

CHANGE:=TRUE;

{ACQUISITION DES DONNEES}

CLRSCR;
WRITE (' ENTREZ LE NOMBRE DE VILLES A VISITER  : ');
READLN (N);
WRITE('');
FOR I:=1 TO N DO
  BEGIN
    WRITE ('ENTREZ LES COORDONNEES DE LA VILLE ');
    WRITE (I);
    WRITE (':  X = ');
    READ (X[I]);
    WRITE ('  Y = ');
    READ (Y[I]);
  END;

{INTIALISATION DE T ET REP}

FOR I:=1 TO N DO
  BEGIN
    FOR J:=1 TO N DO
    BEGIN
      A[I,J]:=SQRT(SQR(X[J]-X[I])+SQR(Y[J]-Y[I]));
    END;
  END;

{CHOIX DU CIRCUIT INITIAL PAS TROP ABSURDE}

C[1]:=1;
FOR I:=1 TO N-1 DO
  BEGIN
    FOR J:=1 TO N DO
      A[J,C[I]]:=0;
```

```
      J:=0;

      REPEAT
            BEGIN
               J:=J+1;
               VAL:=A[C[I],J];
               DMIN:=VAL+1;
            END;

      UNTIL VAL<>0;

      FOR J:=1 TO N DO
         BEGIN
            VAL:=A[C[I],J];
            IF (VAL<>0)AND(VAL<DMIN) THEN
              BEGIN
                 DMIN:=VAL;
                 C[I+1]:=J;
              END;
         END;
   END;

WRITE('CIRCUIT INITIAL=');
FOR I:=1 TO N DO
   BEGIN
      WRITE(C[I]);
      WRITE(' ');
   END;
WRITELN;

{NOUVELLE REALISATION DE LA MATRICE DES LONGUEURS A[ ]}

FOR I:=1 TO N DO
   FOR J:=1 TO N DO
      A[I,J]:=SQRT(SQR(X[J]-X[I])+SQR(Y[J]-Y[I]));
```

```
{CALCUL DE LA LONGUEUR TOTALE DU CIRCUIT INITIAL}

LONG:=A[C[1],C[N]];

FOR I:=1 TO N-1 DO
   LONG:=LONG+A[C[I],C[I+1]];

WRITELN('LONGUEUR INITIALE=');WRITELN(LONG);

{CHOIX DE LA TEMPERATURE INITIALE}

SOMDL:=0;
FOR I:=1 TO REP DO
  BEGIN
    GI:=RANDOM(N-3)+2;
    DI:=RANDOM(N-GI+1)+1;
    IF I=1 THEN DIM1:=N ELSE DIM1:=DI-1;
    IF DI+GI>N THEN DIPGI:=1 ELSE DIPGI:=DI+GI;
    DL:=A[C[DIM1],C[DI+GI-1]]+A[C[DI],C[DIPGI]];
    DL:=DL-A[C[DIM1],C[DI]]-A[C[DI+GI-1],C[DIPGI]];

    IF DL>0 THEN
      BEGIN
        SOMDL:=SOMDL+DL;
        J:=J+1;
      END;
  END;
T:=SOMDL/J/LN(1/0.8);
WRITE('T INITIALE = ');
WRITELN(T);

{TRAITEMENT}

NBITERATION:=0;
WHILE CHANGE DO
    BEGIN
      CHANGE:=FALSE;
```

```
FOR I:=1 TO N DO
COLD[I]:=C[I];
FOR I:=1 TO REP DO
  BEGIN
    GI:=RANDOM(N-3)+2;
    DI:=RANDOM(N-GI+1)+1;
    IF DI=1 THEN DIM1:=N ELSE DIM1:=DI-1;
    IF DI+GI>N THEN DIPGI:=1 ELSE DIPGI:=DI+GI;
    DL:=A[C[DIM1],C[DI+GI-1]]+A[C[DI],C[DIPGI]];
    DL:=DL-A[C[DIM1],C[DI]]-A[C[DI+GI-1],C[DIPGI]];

    IF (DL<=0) THEN
      BEGIN
        CHANGE:=TRUE;
        FOR J:=DI TO TRUNC(DI+GI/2-1) DO
          BEGIN
            Z:=C[J];
            C[J]:=C[2*DI+GI-J-1];
            C[2*DI+GI-J-1]:=Z;
          END;
      END
    ELSE
     BEGIN
      IF DL/T>50 THEN DL:=T*50;
      NBRE:=EXP(-DL/T);
      ALEAT:=RANDOM;
      IF (ALEAT<EXP(-DL/T)) THEN
        BEGIN
          CHANGE:=TRUE;
          FOR J:=DI TO TRUNC(DI+GI/2-1) DO
            BEGIN
              Z:=C[J];
              C[J]:=C[2*DI+GI-J-1];
              C[2*DI+GI-J-1]:=Z;
            END;
        END;
     END;
```

```
            END;

          T:=T*COEFF;
          NBITERATION:=NBITERATION+1;
          WRITE('ITER N° ');
          WRITE(NBITERATION);
          WRITE(' T=');WRITLEN(T);
          IF NBITERATION=STOP THEN
              CHANGE:=FALSE;

{CALCUL DE LA LONGUEUR TOTALE DU CIRCUIT}

LONG:=A[C[1],C[N]];
WRITE ('CIRCUIT : ');
FOR I:=1 TO N-1 DO
   BEGIN
     LONG:=LONG+A[C[I],C[I+1]];
     WRITE(C[I]);
     WRITE(' ');
   END;
WRITELN (C[N]);
WRITE('LONGUEUR TOTALE=');
WRITELN(LONG);
END;

{CALCUL DE LA LONGUEUR TOTALE DU CIRCUIT FINAL}

LONG:=A[C[1],C[N]];
FOR I:=1 TO N-1 DO
   LONG:=LONG+A[C[I],C[I+1]];
WRITE('LONGUEUR FINALE=');WRITELN(LONG);
WRITE ('J AI FINI');
FOR I:=1 TO N DO
  BEGIN
    WRITE (C[I]); WRITE (' ');
  END;
WRITELN;
END.
```

V.2 ALGORITHMES GENETIQUES

Sur base du tableau repris à la figure 5.6, nous pouvons conclure :

PVC à moins de 30 villles

La solution optimale est obtenue en choisissant le nombre de génotypes formant la population M = 30, le taux de croisement PC = 0.25, le taux d'inversion PI=0.4 et le nombre de fois que l'on appllique l'algorithme NT = 5.000. En effet, on obtient des solutions sous-optimales si :

- le paramètre NT est inférieur à 3.000
- le nombre de génotypes M est inférieur à 20
- si le taux de croisement est supérieur à 0.3 ou inférieur à 0.2
- si le taux d'inversion est inférieur à 0.3 ou supérieur à 0.5

PVC à plus de 30 villes

On obtient le circuit le plus court en prenant NT plus grand que 20.000, M plus grand que 25, PC = 0.25 et PI = 0.4.
Plus le nombre de villes augmente, plus il faut augmenter les paramètres NT et M puisque le nombre de circuits possibles augmente considérablement. Pour 50 villes, NT = 25.000 et M = 30 sont une bonne approximation.
En ce qui concerne les paramètres PC et PI, nous tirons les mêmes conclusions que pour les PVC à moins de 30 villes.

La figure 5.7 représente un des 50 circuits initiaux choisi tout à fait arbitrairement pour un PVC à 70 villes et la figure 5.8 décrit le circuit de longueur optimale obtenu après 25.000 itérations.

La figure 5.9 décrit l'évolution de la longueur du circuit en fonction du paramètre NT. Nous constatons que le circuit le plus court est obtenu après 25.000 itérations. Au-delà de cette valeur, aucune amélioration n'est constatée.

Fig 5.6 : TABLEAU DES RESULTATS OBTENUS PAR LA METHODE DES ALGORITHMES GENETIQUES

	NT	M	PC	PI	LONG.
N = 12	1000	30	0.25	0.4	26.24
	5000	5	0.25	0.4	26
	5000	5	0.2	0.3	26
	5000	5	0.15	0.2	26
	5000	10	0.25	0.4	26
	5000	20	0.25	0.4	25.77
	5000	**30**	**0.25**	**0.4**	**25.77**
	5000	30	0.5	0.4	26
	5000	30	0.25	0.9	25.77
	5000	30	0.9	0	40.49
	5000	30	0	0.9	25.77
	5000	30	0.2	0.5	26.24
	5000	30	0.9	0.4	25.77
	20000	5	0.15	0.2	25.77
N = 25	600	30	0.25	0.4	56.81
	1000	10	0.25	0.4	49.06
	1000	30	0.25	0.4	46.21
	1000	30	0.15	0.2	50.68
	1000	30	0.5	0.4	45.20
	2000	30	0.25	0.4	43.11
	5000	10	0.25	0.4	43.25
	5000	**30**	**0.25**	**0.4**	**40.59**
	5000	30	0.5	0.4	42.59
	5000	30	0.15	0.2	42.50
	5000	30	0.9	0	97.07
	5000	30	0	0.9	41.85
	5000	30	0.5	0.9	41.85
	5000	30	0.9	0.9	42.41
	5000	30	0.25	0.9	41.85
	5000	30	0.9	0.4	40.83
	10000	30	0.25	0.4	40.59
N = 50	1000	30	0.25	0.4	89.72
	5000	30	0.25	0.4	65.91
	5000	30	0.5	0.9	63.52
	5000	30	0.9	0	212.4
	5000	30	0	0.9	67.69
	5000	30	0.9	0.4	64.86
	5000	30	0.3	0.5	67.74
	5000	30	0.4	0.6	64.11
	5000	30	0.6	0.4	65.85
	5000	30	0.6	0.9	64.51
	10000	30	0.25	0.4	63.81

	10000	30	0.5	0.9	62.45
	25000	**30**	**0.25**	**0.4**	**60.28**
	25000	30	0.5	0.9	60.52
	25000	30	0.35	0.4	60.63
	25000	30	0.5	0.4	64.04
	25000	30	0.25	0.5	60.69
	25000	30	0.25	0.6	62.38
	25000	30	0.25	0.8	63.45
	25000	30	0.4	0.4	62.69
	25000	50	0.25	0.4	60.63
N = 70	25000	30	0.25	0.4	90.05
	25000	**50**	**0.25**	**0.4**	**87.09**
	25000	50	0.35	0.4	87.75
	25000	50	0.5	0.9	87.22
	25000	50	0.9	0	293.47
	25000	50	0	0.9	97.32
	25000	50	0.5	0.4	92.63
	25000	50	0.25	0.8	91.12

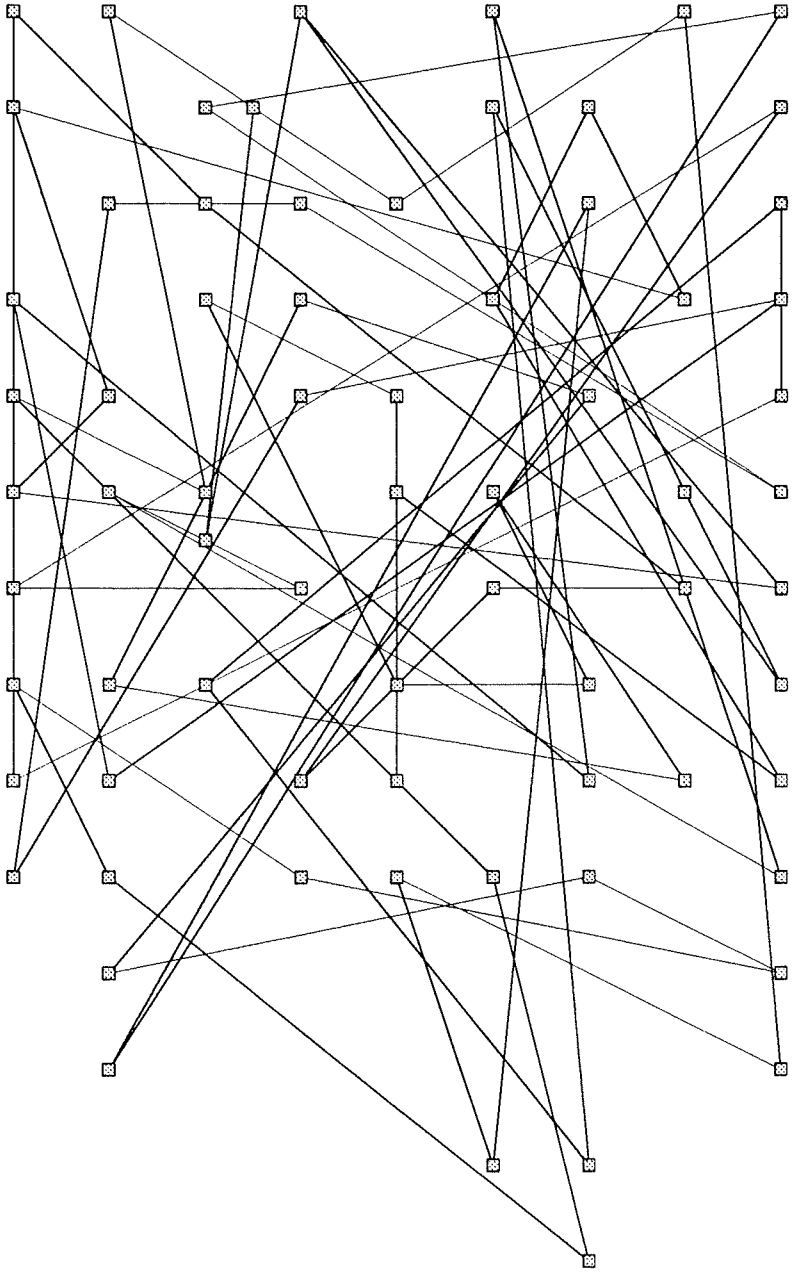


Fig.5.7

Algorithmes génétiques : circuit initial

N = 70 villes
PC = 0.25
PI = 0.4
M = 50
NT = 25000

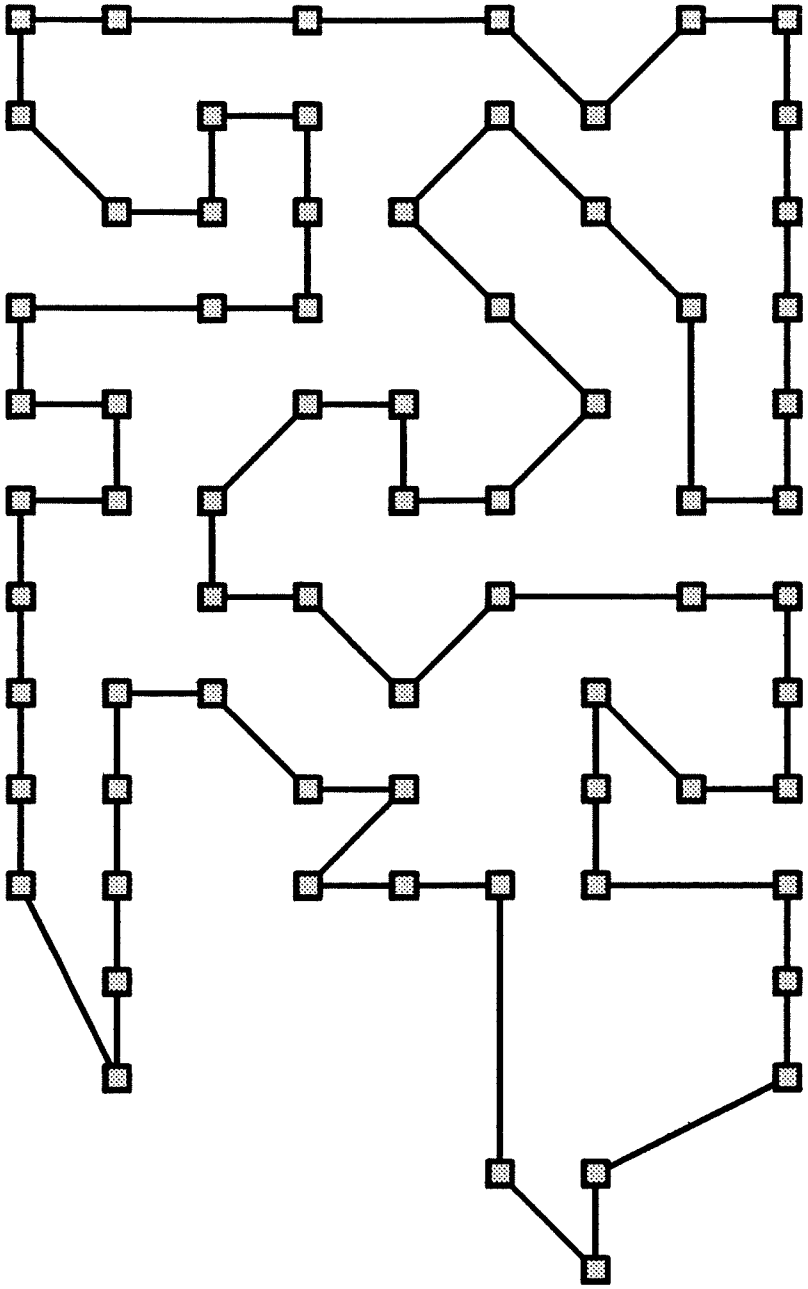


Fig.5.8

Algorithmes génétiques : circuit final

N = 70 villes
PC = 0.25
PI = 0.4
M = 50
NT = 25000
LONGUEUR = 87.09

LISTING DU PROGRAMME "ALGORITHMES GENETIQUES"

```
PROGRAM ALGO_GEN;

USES CRT;

TYPE        TAB=ARRAY[1..70] OF ARRAY[1..70] OF REAL;
            MAT=ARRAY[1..50] OF ARRAY[1..70] OF INTEGER;
            VECT=ARRAY[1..70] OF INTEGER;
            VECT2=ARRAY[1..50] OF INTEGER;
            VECTREEL=ARRAY[1..50] OF REAL;

CONST       NT=25000;

VAR         H,NB,K,I,J,N,Z,GI,DI,DL,DMIN,LONG,REP,VAL : INTEGER;
            M,NA,T,PG,PG1,PG2,PP,PPRIM : INTEGER;
            GENX,GENY,PK,INTER : INTEGER;
            A : TAB;
            X,Y,NO : VECT;
            OK,CROISER,MODIF : BOOLEAN;
            LTINV,LTOT,SOMP,P,PC,PI : REAL;
            B,BPRIM : MAT;
            COPIE : VECT2;
            L,ALEAT,LPRIM : VECTREEL;
            COORDXY : FILE OF INTEGER;


BEGIN

{* INTRODUCTION DES PARAMETRES CONSTANTS *}

CLRSCR;
WRITELN('NT = 25000');
WRITELN;
WRITE('ENTREZ M ');
READLN(M);
```

```
WRITELN;
WRITE('ENTREZ PC ');
READLN(PC);
WRITELN;
WRITE('ENTREZ PI ');
READLN(PI);

{INITIALISATION}
PK:=0;
RANDOMIZE;

FOR I:=1 TO 90 DO
  BEGIN
    FOR J:=1 TO 90 DO
      A[I,J]:=0;
    X[I]:=0;
    Y[I]:=0;
    NO[I]:=0;
    FOR J:=1 TO M DO
      BEGIN
          B[J,I]:=0;
          BPRIM[J,I]:=0;
      END;
  END;
FOR I:=1 TO M DO
      BEGIN
          L[I]:=0;
          ALEAT[I]:=0;
         COPIE[I]:=0;
      END;


{ACQUISITION DES DONNEES}

WRITE (' ENTREZ LE NOMBRE DE VILLES A VISITER  : ');
READLN (N);

{****LECTURE DES N COORDONNEES DANS FICHIER COORDXY****}
```

```
 {$I-}
 ASSIGN(COORDXY,'VILLES.DTA');
 IF IORESULT<>0 THEN WRITELN('ERREUR FICHIER');
 RESET(COORDXY);
 IF IORESULT<>0 THEN WRITELN('ERREUR FICHIER');
 {$I+}
 FOR I:=1 TO N DO
   BEGIN
      READ(COORDXY,X[I]);
      READ(COORDXY,Y[I]);
      WRITELN('VILLE N°',I,'  X=',X[I],'  Y=',Y[I]);
   END;
 CLOSE(COORDXY);

{INTIALISATION DE T ET REP}

T:=0;
FOR I:=1 TO N DO
  BEGIN
    FOR J:=1 TO N DO
    BEGIN
      A[I,J]:=SQRT(SQR(X[J]-X[I])+SQR(Y[J]-Y[I]));
      WRITE(A[I,J]);
    END;
    WRITELN('');
  END;

{CHOIX AU HASARD DE M GENOTYPES POUR FORMER LA
POPULATION INITIALE}

   FOR I:=1 TO N DO
     BEGIN
       FOR J:=1 TO M DO
         BEGIN
            NA:=RANDOM(N)+1;
            MODIF:=FALSE;
            REPEAT
```

```
            IF B[J,NA]=0 THEN
              BEGIN
                B[J,NA]:=I;
                MODIF:=TRUE;
              END;
              IF NA<N THEN
              NA := NA+1
              ELSE NA := 1;
          UNTIL MODIF;
        END;
    END;
FOR T:=0 TO NT DO
BEGIN

(*MESURE DE LA LONGUEUR*)

   FOR J:=1 TO M DO
     BEGIN
       L[J]:=0;
       FOR I:=1 TO N-1 DO
         L[J]:=L[J]+A[B[J,I],B[J,I+1]];
       L[J]:=L[J]+A[B[J,N],B[J,1]];
     END;

   (*CALCUL PROBABILITES*)

   LTINV:=0;
   LTOT:=0;
   FOR J:=1 TO M DO
     BEGIN
       LTINV:=LTINV+1/L[J];
       LTOT:=LTOT+L[J];
       ALEAT[J]:=RANDOM;
     END;

   SOMP:=0;

   FOR J:=1 TO M DO
```

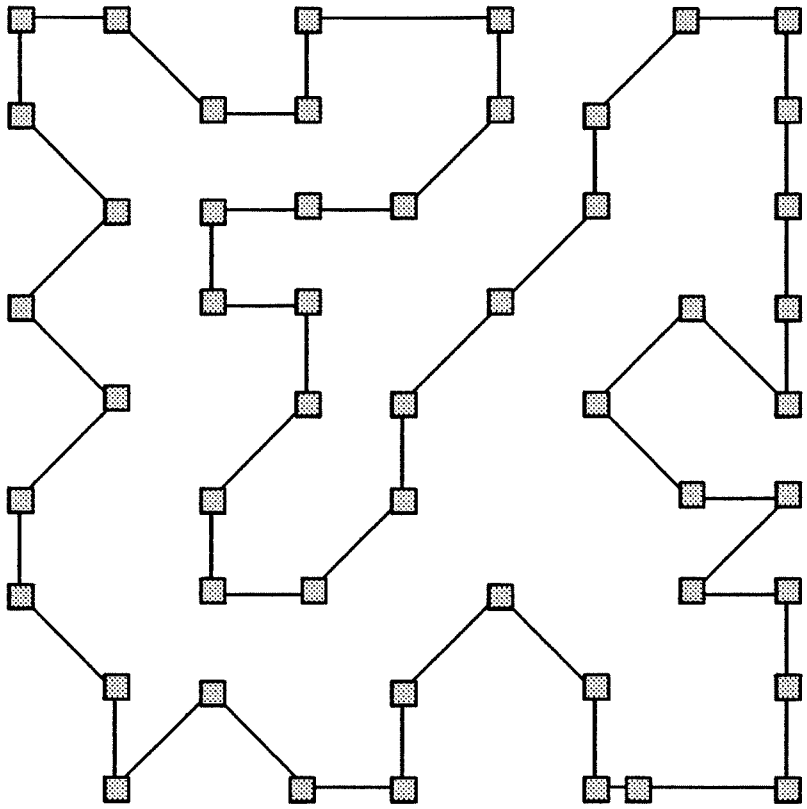
```
  BEGIN
   P:=(1/L[J])/LTINV;
   NB:=0;
   FOR K:=1 TO M DO
     BEGIN
       IF (ALEAT[K]>SOMP) AND (ALEAT[K]<=SOMP+P) THEN
         NB:=NB+1;
     END;

   SOMP:=SOMP+P;
   COPIE[J]:=NB;
  END;

(*REPRODUCTION*)

H:=0;
FOR J:=1 TO M DO
  BEGIN
    IF COPIE[J]<>0 THEN
      FOR K:=1 TO COPIE[J] DO
        BEGIN
          H:=H+1;
          FOR I:=1 TO N DO
            BEGIN
              BPRIM[H,I]:=B[J,I];
            END;
            LPRIM[H]:=L[J];
        END;

  END;

  FOR J:=1 TO M DO
    BEGIN
        FOR I:=1 TO N DO
              B[J,I]:=BPRIM[J,I];
              L[J]:=LPRIM[J];
        END;
```

```
(*CROISEMENT*)

FOR K:=1 TO TRUNC(M*PC) DO
  BEGIN
    REPEAT
      GENX:=RANDOM(M)+1;
      GENY:=RANDOM(M)+1;
    UNTIL (GENX<>GENY);

    GI:=RANDOM(N-3)+2;
    DI:=RANDOM(N-GI+1)+1;
    FOR I:=DI TO DI+GI-1 DO
      BEGIN
        J:=0;
        REPEAT
          J:=J+1;
        UNTIL B[GENX,I]=B[GENY,J];
        NO[I-DI+1]:=J;
      END;
    PG:=0;
    FOR I:=1 TO GI DO
      IF PG<NO[I]
        THEN
        PG:=NO[I];
    PG1:=PG;

    CROISER:=TRUE;

    FOR I:=1 TO GI-1 DO
      BEGIN
        PG:=PG-1;
        OK:=FALSE;
        FOR J:=1 TO GI DO
          IF NO[J]=PG THEN OK:=TRUE;
          IF OK=FALSE THEN CROISER:=FALSE;
      END;
```

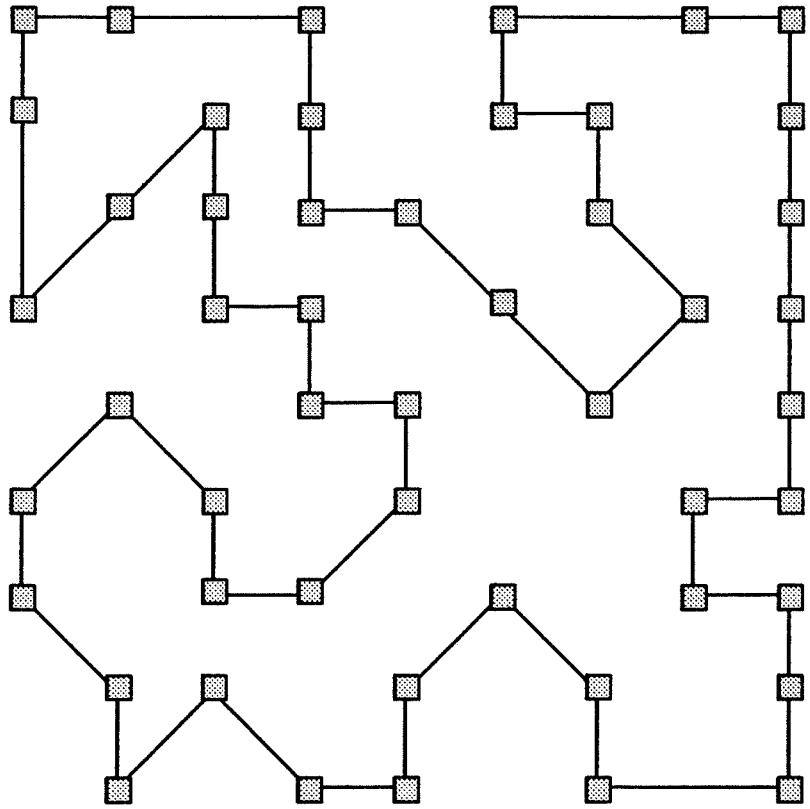
```
IF CROISER THEN
  BEGIN
    PG2:=PG1;
    FOR I:=1 TO N DO
      BEGIN
        BPRIM[GENX,I]:=B[GENX,I];
        BPRIM[GENY,I]:=B[GENY,I];
      END;
    FOR I:=DI TO DI+GI-1 DO
      BEGIN
        PG2:=PG2+1;
        INTER:=BPRIM[GENY,PG2-GI];
        BPRIM[GENY,PG2-GI]:=BPRIM[GENX,I];
        BPRIM[GENX,I]:=INTER;
      END;
    LPRIM[GENX]:=0;
    LPRIM[GENY]:=0;
    FOR I:=1 TO N-1 DO
      BEGIN
        LPRIM[GENX]:=LPRIM[GENX]+
                        A[BPRIM[GENX,I],BPRIM[GENX,I+1]];
        LPRIM[GENY]:=LPRIM[GENY]+
                        A[BPRIM[GENY,I],BPRIM[GENY,I+1]];
      END;
    LPRIM[GENX]:=LPRIM[GENX]+
                         A[BPRIM[GENX,N],BPRIM[GENX,1]];
    LPRIM[GENY]:=LPRIM[GENY]+
                         A[BPRIM[GENY,N],BPRIM[GENY,1]];
    IF LPRIM[GENX]<LPRIM[GENY] THEN PPRIM:=GENX
                  ELSE PPRIM:=GENY;
    IF L[GENX]<L[GENY] THEN PP:=GENX
                  ELSE PP:=GENY;
    IF LPRIM[PPRIM]<L[PP] THEN
      BEGIN
        FOR I:=DI TO DI+GI-1 DO
        BEGIN
          PG1:=PG1+1;
          INTER:=B[GENY,PG1-GI];
```

```
            B[GENY,PG1-GI]:=B[GENX,I];
            B[GENX,I]:=INTER;
           END;
            L[GENX]:=LPRIM[GENX];
            L[GENY]:=LPRIM[GENY];
         END;
     END;
  END;

(*INVERSION*)

FOR K:=1 TO TRUNC(M*PI) DO
  BEGIN
    GENX:=RANDOM(M)+1;
    GI:=RANDOM(N-3)+2;
    DI:=RANDOM(N-GI+1)+1;
    FOR I:=1 TO N DO
     BPRIM[GENX,I]:=B[GENX,I];
    FOR J:=DI TO TRUNC(DI+GI/2-1) DO
    BEGIN
      INTER:=BPRIM[GENX,J];
      BPRIM[GENX,J]:=BPRIM[GENX,2*DI+GI-J-1];
      BPRIM[GENX,2*DI+GI-J-1]:=INTER;
    END;
    LPRIM[GENX]:=0;
    FOR I:=1 TO N-1 DO
       LPRIM[GENX]:=LPRIM[GENX]+
                              A[BPRIM[GENX,I],BPRIM[GENX,I+1]];
      LPRIM[GENX]:=LPRIM[GENX]+
                               A[BPRIM[GENX,N],BPRIM[GENX,1]];
      IF LPRIM[GENX]<=L[GENX] THEN
        BEGIN
          FOR J:=DI TO TRUNC(DI+GI/2-1) DO
            BEGIN
              INTER:=B[GENX,J];
              B[GENX,J]:=B[GENX,2*DI+GI-J-1];
              B[GENX,2*DI+GI-J-1]:=INTER;
            END;
```

```
                L[GENX]:=LPRIM[GENX];
              END;
          END;
        WRITE(PK);
        PK:=PK+1;
      END;
    WRITE ('J AI FINI');
    FOR I:=1 TO M DO
     BEGIN
       L[I]:=0;
       FOR J:=1 TO N-1 DO L[I]:=L[I]+A[B[I,J],B[I,J+1]];
       L[I]:=L[I]+A[B[I,N],B[I,1]];
     END;
    FOR I:=1 TO M DO
     BEGIN
      FOR J:=1 TO N DO
        BEGIN
            WRITE(B[I,J]);
            WRITE(' ');
        END;
       WRITE ('LONG = ');
       WRITE (L[I]);WRITE (' ');
  END;

END.
```



Algorithmes génétiques : PVC à 50 villes
NT = 25000 M = 30 PC = 0.25 PI = 0.4
LONGUEUR = 60.28



Recuit simulé : PVC à 50 villes
COEFF = 0.93 REP = 5000
LONGUEUR = 60

CONCLUSION

Les différentes expériences qui ont été réalisées nous ont montré l'aptitude des algorithmes génétiques et du recuit simulé à maîtriser des systèmes dynamiques complexes. L'application de ces deux techniques au problème du voyageur de commerce nous a permis de déterminer avec précision, les divers paramètres de contrôle à considérer pour obtenir les meilleures performances.

Dans la pratique, il faut souvent faire un compromis entre le temps de traitement qui se chiffre en heures et la qualité des résultats obtenus. C'est pourquoi, on se contente bien souvent de trouver une solution "acceptable" et l'on cherche rarement à atteindre l'optimum absolu.

Nous avons pu constater que ces deux méthodes apportent des solutions identiques au PVC et ce, quelle que soit la complexité du problème. Toutefois, la technique du recuit simulé s'avère plus facile à implémenter vu qu'elle fait appel à moins de paramètres de contrôle et qu'elle ne se compose que d'une seule étape de traitement. Nous avons cependant simplifié l'implémentation des algorithmes génétiques de plusieurs façons. En premier lieu, nous avons négligé d'autres types de stratégies de sélection et d'autres types d'opérateurs tels que le croisement multipoint et la mutation. En second lieu, les AG que nous considérons sont essentiellement des procédures d'optimisation sans contraintes. Nous avons vu précédemment qu'il existe différentes manières d'incorporer des contraintes dans les AG. Toutefois, l'effet de ces contraintes sur le résultat final n'a pas encore été établi avec exactitude. En ce qui concerne le temps d'exécution, le traitement du recuit simulé semble plus rapide que la technique des AG. Notons cependant que le recuit simulé nécessite une configuration initiale pas trop absurde tandis que les AG partent d'une configuration initiale totalement aléatoire.

Une autre manière d'implémenter les AG serait de permettre à ceux-ci d'ajuster automatiquement la valeur de leurs paramères pendant la recherche. Cette méthode n'est pas utilisée car elle nécessiterait un temps de traitement trop long étant donné que la valeur initiale des paramètres ne serait pas appropriée à la complexité du problème. Par conséquent, les expériences décrites ci-dessus sont indispensables pour identifier les paramètres de contrôle conduisant aux solutions optimales.

ANNEXE 1 : RAPPEL DE BIOLOGIE

La génétique est la science qui étudie l'hérédité.

L'hérédité est l'ensemble des caractères transmis des parents aux descendants.

Les chromosomes contiennent les gènes qui déterminent les caractères héréditaires et qui sont composés de quelques milliers de paires de bases (les bases étant des substances qui, combinées avec un acide, produisent un sel et de l'eau). Ils sont en forme de filaments ou de bâtonnets et sont en nombre constant (et pair) dans toutes les cellules d'un même individu et chez tous les individus de la même espèce.

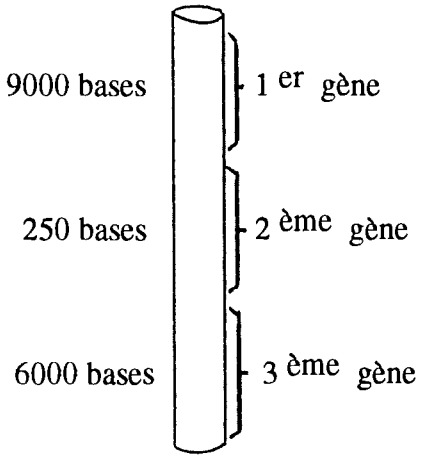


Figure 1 : Un chromosome

On parle d'allèle pour désigner un caractère héréditaire qui s'oppose à un autre, par exemple des graines de pois lisses ou ridées.

Croisement entre deux parents

Considérons l'exemple des pois sur lesquels reposent les premiers travaux de la génétique. Le caractère étudié sera la longueur de la tige.

Soit G le gène dominant responsable de la tige géante (dominant signifie masquer l'autre gène).

Soit g le gène récessif responsable de la tige naine.

Le croisement de deux parents Gg donne :

	G	g
G	GG	Gg
g	Gg	gg

Le génotype d'un individu reprend tous les gènes de cet individu. Il s'agit d'une paire de symboles utilisés pour représenter les gènes responsables d'un caractère donné. Dans l'exemple ci-dessus, le génotype sera (GG,Gg,gg). On écrira plus souvent (1,2,1) pour dire qu'il existe un génotype géant de race pure (GG), deux génotypes géants hybrides (Gg) et un génotype nain de race pure (gg).

Le phénotype est le résultat de la présence des gènes chez un individu; c'est ce que l'on observe. Dans l'exemple, le phénotype sera (3,1) et signifie qu'il existe deux espèces : l'espèce à tige géante (GG,Gg,Gg) et l'espèce à tige naine (gg).

Liaison des gènes (linkage) et crossing-over

Un chromosome porte un grand nombre de gènes; ces gènes sont liés les uns aux autres dans un arrangement linéaire. Cet arrangement porte le nom de linkage. La liaison entre les gènes n'est cependant pas toujours parfaite. En effet, des segments correspondants de chromosomes homologues peuvent se détacher et être échangés : c'est le phénomène de crossing-over.

Imaginons des gènes A, B, C fixés sur une chromatide et les gènes a, b, c fixés sur une chromatide du chromosome homologue; le crossing-over donne des nouvelles combinaisons de gènes qui n'étaient pas présentes chez les parents.

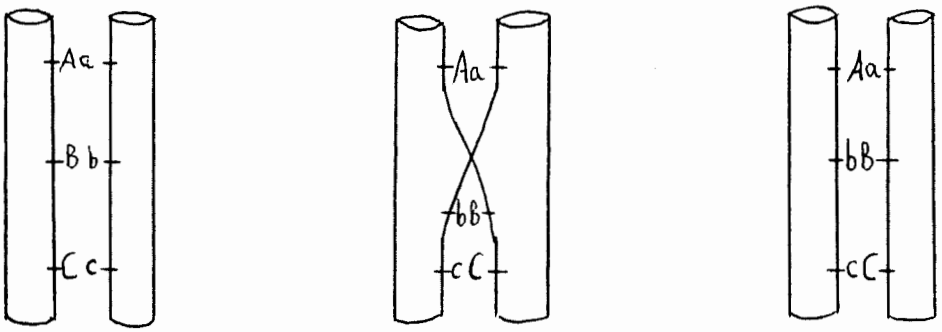


Figure 2 : Crossing-over

ANNEXE 2 : DISTANCE DE HAMMING

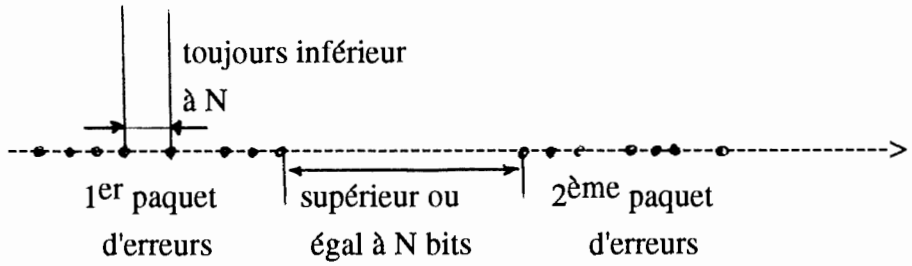
Nous savons que toute information binaire émise par une source émettrice vers un récepteur transite au travers d'une voie appelée voie de transmission.

Une des caractéristiques d'une liaison de transmission de données est le taux d'erreurs

$$t = \frac{\text{nombre de bits reçus erronés}}{\text{nombre total de bits transmis}}$$

STRUCTURE DES ERREURS

Il est apparu nécessaire de définir la notion de paquet d'erreurs. N étant un nombre fixé à l'avance (en général pris égal à 10), un paquet d'erreurs est une série de bits dans laquelle le nombre de bits exacts, séparant deux bits erronés est toujours inférieur à N. Entre le dernier bit erroné d'un paquet d'erreurs et le premier bit erroné du paquet suivant, on trouve évidemment au moins N bits exacts.



Des messages erronés correspondent à une suite binaire formée en additionnant, modulo 2, les chiffres de la suite émise et une suite qui représente la voie.

1 0 1 1 0 1 1 0 0 0	suite émise
+0 0 1 0 0 0 0 0 1 0	mot "erreur" introduit par la voie
1 0 0 1 0 1 1 0 1 0	suite reçue

Les propriétés habituelles de l'addition modulo 2, à savoir

$$0 + 0 = 0$$

$1 + 0 = 1$

$0 + 1 = 1$

$1 + 1 = 0$

indiquent simplement que chaque élément binaire erroné peut être considéré comme résultant de l'addition modulo 2 de cet élément lui-même et d'un 1 (chaque élément exact résulte de l'addition de l'élément et d'un 0). La perturbation introduite sur le message dépend directement du nombre de 1 du mot "erreur". Le poids d'un mot est le nombre de 1 qu'il contient et la distance de Hamming de deux mots est le poids de leur différence ou, ce qui revient au même dans l'algèbre modulo 2, de leur somme, c'est-à-dire le nombre de bits de même rang par lequel ils diffèrent. Ainsi, les deux mots 110110 et 111010 sont à une distance 2. Autrement dit, un mot émis et un mot reçu sont à une distance égale au poids du mot erreur introduit par la voie. Cette notion est surtout intéressante si l'on considère les mots d'un code. Deux mots seront d'autant plus faciles à distinguer que leur distance de Hamming sera plus grande. En effet, si cette distance est d, il faut d erreurs convenablement placées pour transformer un mot dans l'autre. Pour chaque code, il est donc intéressant d'examiner quelle est la distance de Hamming minimale qui peut être rencontrée, la distance nulle correspond à l'identité de deux mots étant évidemment exclue.

BIBLIOGRAPHIE

[1] **GENETIQUE,** Robert P. Levine, McGraw Hill

[2] **GENETIQUE**, cours et problèmes, série Schaum

[3] **GENETIC ALGORITHMS**, Introduction, Applications and Extensions, Piet Spiessens, A.I. MEMO n° 88-19, Vrije Universiteit Brussel

[4] **OPTIMIZATION OF CONTROL PARAMETERS FOR GENETIC ALGORITHMS**, Greffenstette John J., IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-16, n° 1 - january/february 1986

[5] **GENETIC ALGORITHMS AND SIMULATED ANNEALING**, Davis Lawrence, Morgan Kaufmann, Los Altos, CA (1987)

[6] **THE EVOLUTION OF STRATEGIES IN THE ITERATED PRISONER'S DILEMMA**, Robert Axelrod (1987)

[7] **INCORPORATING PROBLEM SPECIFIC KNOWLEDGE INTO GENETIC ALGORITHMS**, Grefenstette John J. (1987)

[8] **IMPROVING SEARCH IN GENETIC ALGORITHMS**, Booker Lashon (1987)

[9] **SIMULATED ANNEALING ALGORITHM FOR THE MINIMUM WEIGHTED PERFECT EUCLIDEAN MATCHING PROBLEM,** R.A.I.R.O. - Recherche opérationnelle / Operations Research, vol 20, n° 3 - août 1986, pages 177 à 197

[10] **LE RECUIT SIMULE**, Jean-Luc Lutton et Ernesto Bonomi, Pour la Science n°129 - juillet 1988

[11] **THE N-CITY TRAVELLING SALESMAN PROBLEM : STATISTICAL MECHANICS AND THE METROPOLIS ALGORITHM**, SIAM Review, vol. 26 n° 4 - octobre 1984, p. 551 à 568

[12] **CHAOS ET C.A.O. OU LA METHODE DU RECUIT SIMULE,**
G. Dreyfus, Afcet / Interfaces n° 53 - mars 1987

[13] **GRAPHES ET ALGORITHMES**, Michel Gondram et M. Minoux,
Editions Eyrolles

[14] **ENCYCLOPEDIE DES SCIENCES INDUSTRIELLES QUILLET,**
Librairie Aristide Quillet - Paris VII^{e}